

# 1-概述

唯一不因项目而异

复杂性	本 质 和 基 础 理 念
不可见性	
可变性	
一致性	

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，从而导致软件开发和维护过程中出现一系列严重问题的现象

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

管理视角：能否复制成功？

技术视角：是否能将问题解决得更好？

广义软件过程 = 技术 + 人员 + 框架 + 过程

两种开发方法：软件开发过程

XP SCRUM 敏捷  
Gate Cleanroom 质量/质量

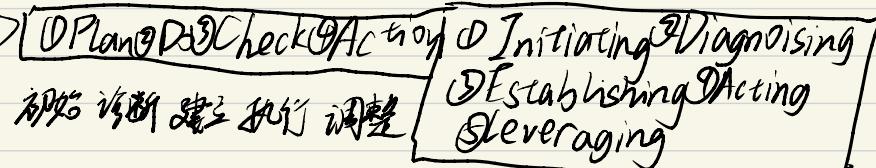
管理的目的是为了让软件过程在开发效率、质量等方面有更好性能

瀑布模型  
迭代模型  
增量模型  
螺旋模型  
原型法

生命周期模型  
一种人为划分  
主框架 不包括技术实践  
一种组织度划分

软件过程管理参考模型 CMM/CMMI, SPICE

软件过程改进参考元模型 PDCA IDEAL →



项目管理：通常关注特定项目及其目标的实现

过程管理：关注团队内所有项目，它使用的方法有什么提升

1.2.1.1 软件管理

目标 管理关键要素  
状态  
偏差  
改进

成本 日  
质量 周期

软件项目管理？

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

## 2- 软件过程的历史演变和经典工作 -1

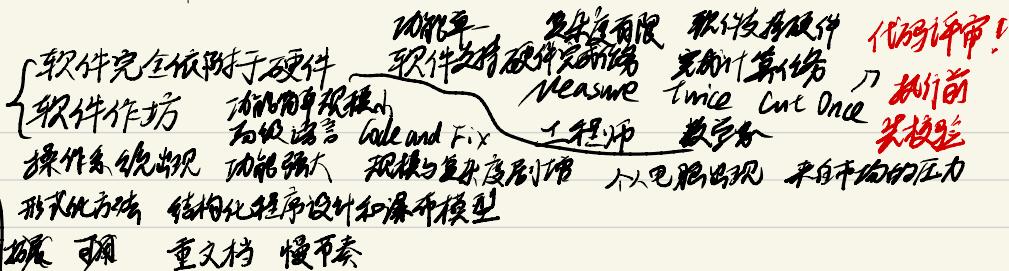
① 软硬件一体化阶段 50年代-70年代

② 软件成为独立的产品 70年代-90年代

③ 网络化和服务化 90年代中期至今

功能多，规模大，用户急剧增加  
快速变化，不确定，需求多样化

CMM&CMMI



CMM2 需求管理 软件项目计划

当前情况 如何 软件项目跟踪与监控

是否需要干预 软件质量管理 软件配置管理

CMM3 组织过程重点

组织过程定义

集成软件管理

软件产品工程

组织培训 识别相同

同行评审

CMM4 疏散预防 技术改革管理

过程变更管理

CMM5 定量过程管理 软件质量管理

CMMI 相比 CMM, 集成更易模型进入,

如系统工程能力整合能力等?

而不是过程优秀标准, 不比较能力

不是软件过程/软件研究方法 教授是

CMMI 初始级 Initial Initial

CMMI2 可重复级 Repeatable Managed

CMMI3 已定义级 Defined Defined

CMMI4 已管理级 Managed Quantitatively, Managed 定量管理级

CMMI5 优化级 Optimizing Optimizing

个人英雄主义  
错过机会  
小组级项目管理

有项目计划与资源  
制定很弱的项目特点  
没有专属开发过程

CMMI1 初级

CMMI2 已管理级

CMMI3 已定义级

CMMI4 优化级

CMMI5 改进级

是否需要干预

组织过程重点

维持现状能力建立?

如何调整?

什么结果?

对偏差的理解决策

同行评审

当前情况 如何 软件项目跟踪与监控

是否需要干预 软件质量管理 软件配置管理

CMM3 组织过程重点

组织过程定义

集成软件管理

软件产品工程

组织培训 识别相同

同行评审

CMM4 疏散预防 技术改革管理

过程变更管理

CMM5 定量过程管理 软件质量管理

CMMI 相比 CMM, 集成更易模型进入,

如系统工程能力整合能力等?

而不是过程优秀标准, 不比较能力

不是软件过程/软件研究方法 教授是

CMMI2 需求管理 项目计划、项目监督和控制、对偏差的理解决策

供应商合同管理 产品过程质量保证

配置管理 测量分析

需求开发 技术解决方案、产品集成验证

确认、组织过程重点、组织过程定义

组织的一体化环境

组织的过程性能 定量项目管理

原因分析和解决方案、组织创新和部署

过程表达：连接式

过程管理

项目管理

工程

支持

CMMI3

需求开发 技术解决方案、产品集成验证

确认、组织过程重点、组织过程定义

组织的一体化环境

组织的过程性能 定量项目管理

原因分析和解决方案、组织创新和部署

CMMI4

CMMI5

## 2-软件工程的历史演变和经典工作 - 2

XP - Extreme Programming 工程实践描述

- 快速迭代、增量开发、验收测试 ... 很多很多

## SCRUM 管理框架与管理实践

- 1) 产品订单: product backlog 整个项目的概要文档 包含已规划等级的项目要开发的系统或产品的需求清单, 包括功能和非功能需求以及其他假设和约束条件。 动态调整 改变 可用性 实用性 竞争性
- 2) 冲刺订单: sprint backlog 细化了文档 团队如何实现下一个冲刺  
需求 每一个任务不超过 16 h 以 h 为单位
- 3)燃尽图 —— 增加会议 阶段会议 评审 回顾 冲刺

团队组成: 产品负责人 开发团队 SCRUM Master

跨职能的自组织团队

- ↳ 管理产品待办事项 负责人 开发团队开发产品价值最大化
- ↳ Sprint 线程支持增量可交付已完成增量 自己组织自己管理
- 服务, 支持, 理解 维持 SCRUM 过程

SCRUM 和 TDD 都可以进阶迭代 & 演进  
行为激励靠反馈而非马斯洛层次理论

## 敏捷宣言

个体和互动高于流程与文档  
可工作的软件高于详尽的文档

客户合作胜过合同谈判  
响应变化胜过遵循计划

尽管有可用价值, 我们更侧重于可持续价值

SPICE  
Software Process Improvement and Capability Determination  
工程过程 客户供应高过程 管理过程 支持过程 组织过程

## DevOps

敏捷软件开发  
方法论基础 精益思想  
看板方法

Docker 容器驱动设计的部署的微服务架构  
(大量虚拟机技术使用)

① IaaS

② 工具链与自动化

# 3 团队动力学 自由程与 TSP SCRUM 第一章

软件开发：智力劳动 / 处理和讨论较为抽象概念  $\rightarrow$  全身心参与  
 软件开发：整合完整系统  $\rightarrow$  主观意愿追求卓越

激励手段  $\left\{ \begin{array}{l} \text{激励阶段} \\ \text{维持激励阶段} \end{array} \right.$   
 管理者无法管理工作者  
 知识工作者必须实现且学会自我管理  
 领导者，而非经理

威逼  $\left\{ \begin{array}{l} \text{奖励} \\ \text{惩罚} \end{array} \right.$  马斯洛层次理论

鼓励承诺 - 奖励型

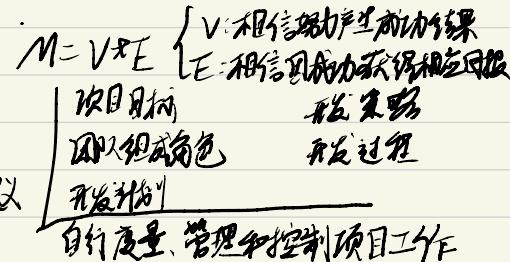
所有团队成员共同参与作出承诺  
 团队依赖于每一位成员履行自己的承诺

自信  
公开  
可信  
向团队承诺

海兹伯格激励理论

麦克勒格 X-Y 理论 Y 理论  
 期望理论  $\rightarrow$  在后训练

## 期望理论



外部环境：项目启动阶段与项目进展中 管理层支持  
 定期维护 需求计划 跟踪进展之表  
 定期汇报 ...

## PSP 团队项目 TSP TSP

个人技能培养  
① 个体经验程度  
② 过程规范  
③ 估算与计划  
④ 质量管理  
⑤ 团队构建过程  
⑥ 团队工作过程  
⑦ 团队交流回顾  
⑧ 团队跟踪  
⑨ 团队角色  
⑩ 团队进展报告  
⑪ 团队会议  
⑫ 团队冲突解决  
⑬ 团队领导

进阶的 + 支持性工作范围：很高级项目  
 创造力想象力 确认信心 组织目标  
 自我约束 自我导向与控制 局限承担责任  
 马斯洛的高层次需求（尊重和自我实现）进行激励

维持激励水平：反馈  
 内在：激励因素  
 成就感 责任感 提升 资源认可  
 外在：保健因素：工作环境 领导 工作关系 安全  
 不喜欢、逃避工作  
 缺乏进取心 没有解决问题与创造问题能力 Y  
 高级辅导：避免承担责任，缺乏主动性  
 自我中心 对组织富有反应冷漠 反应差  
 团队高层互动

### 3- 团队动力学

#### TSP Launch

第一次会议：建立产品目标和业务目标

第二次会议：角色分配和小组目标定义

第三次会议：开发流程定义与策略选择

第四次会议：整体计划

第五次会议：质量计划

第六次会议：个人计划以及计划平衡

第七次会议：风险评估

第八次会议：准备向管理层汇报计划

第九次会议：向管理层汇报计划内容

stakeholder team coach

规模 环境 目标 技能

整体步骤

排日程+估算 潜在冲突的调

整活动程度？与整体计划匹配

整体进度和各里程碑同时完成

来源 变动 应对 数量问题

准备

呼应

项目组长

计划经理

开发经理

质量经理

过程经理

支持经理 和工具

开发人员

经理 vs 领导者

告知 倾听

指导 询问

说明 激励/挑战

决定 促进达成致

控制 教练

监控 授权

设定目标，挑战

Launch 阶段总结

沟通机制有效性  
估算和计划部分  
安排

一个团队必须至少包括两个成员，他们为了共同的目标和愿景努力工作，他们每个人都  
明确的角色和相应的职责定义，任务的完成需要团队成员互相依赖与支持

# 4 - 估算、计划与跟踪 —— 估算

估算：提供决策依据 估算对象：规模、时间与日程

历史数据获取、度量

① 估算是否现猜测 ② 能力如何提升 ③ 用不用模型及好坏

① 精确度量方式往往不便于早期规划/估算 ② 有助于早期规划，但精度往往不能产生精确结果

度量体现决策者对项目实现目标的  
主观程度

GQM GQM

PROBE: PROxy Based Estimation 精确度量与早期规划间桥梁（指成一个组件估算）

概要设计

代理识别和代理规模



元素、规模

没有跟着设计

PSP基本度量项

估算并调整程序规模

估算并调整资源

就没有计划

规模 时间 缺陷 日程

计算预测区间

计算预测区间

整合多个估算结果

① 整合个人多次

② 整合个人估算结果

时间日志、缺陷日志

① 尽可能划分详细  $\pm 50\%$  100h

幼稚回归 加加 平方根区间

所属阶段

② 建立对结果的信心

$25 \text{ 部件} \rightarrow \sigma = 100 \text{ 范围 } 900-1100$

序号

③ 依赖数据

SCRUM: 度量一个 story 的工作量

开始时间

④ 估算要的是过程，而非结果

抽象

发现日期

估算不是相干关系人达成一致的过程

共识 信心 详细 数据 预测

结束时间

注入阶段

中断时间

消除阶段

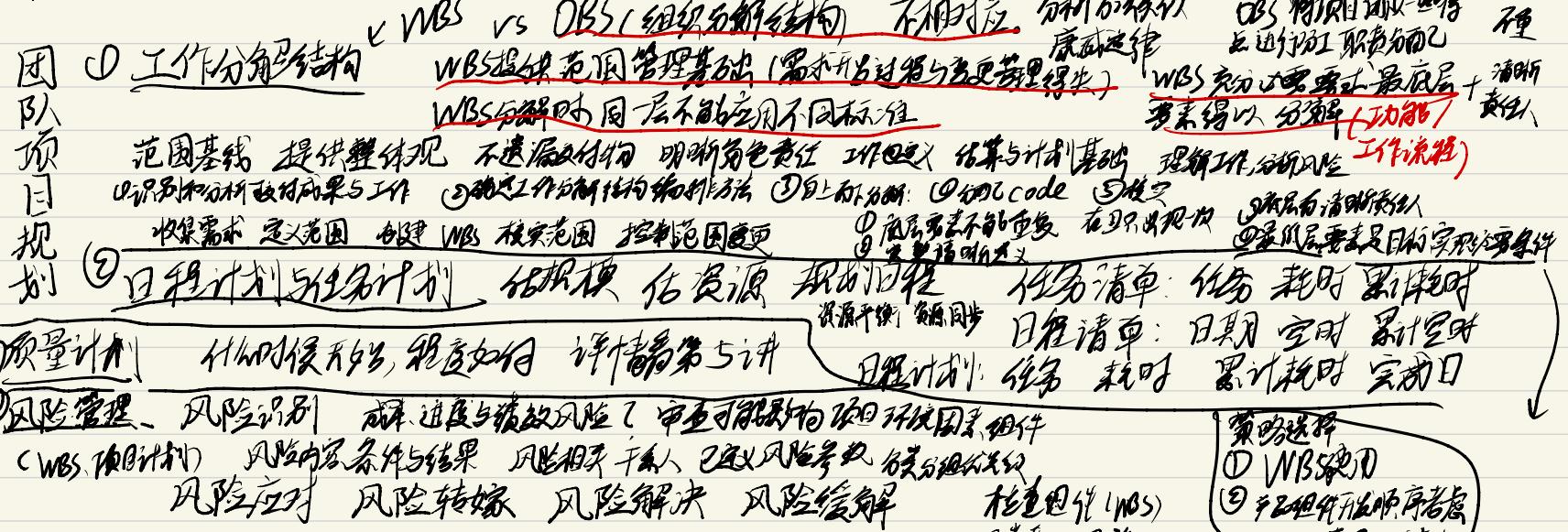
争时间

消除时间

关联缺陷

简要描述

# 4-估算、计划和跟踪 —— 计划



## 计划评审和各方承诺 获取承诺 审查承诺

- 识别每一项计划所需支持, 并与相关干系人沟通承诺
- 记录所有的承诺, 包括完整的与临时的, 并确保由谁负责谁需
- 适当的资源管理人员审查承诺

检查图件 (WBS)  
分解 - 风险  
系统审查会议  
高层级项目计划  
检查已知报告  
检查风险登记

识别的计划  
干系人会议

## ①②③④制定计划 基于历史数据 变量量化化 “无底洞绝对”

# 4 - 估算、计划和跟踪 — 跟踪

了解进度，纠偏

挣值管理

EVM

进度计划

蓝色

Earned Value

成本预算 (BAC, 绿色)

Budget at Completion

Actual Cost

EV, 绿色

SV = PV - EV

CV = AC - EV

AC, 红色

实际成本 (AC, 红色)

— 红色

两种方法：CP图 (倒过来画)

- ① EVM 不能应用软件项目的质量管理
- ② EVM 需要定量化的管理机制在探索型项目
- ③ 挣值管理 完全依赖精确估算但在项目早期起驱动

里程碑评审 在某时间点，标记某项工作完成所见结果

- ① 项目相关承诺 日期 规格 质量
- ② 项目各项执行计划的执行情况
- ③ 项目当前状态讨论
- ④ 项目面临风险讨论

高一级包括低等级内容

EV = CPI \* BAC 成本无偏差

SPI = EV / PV 工程偏差指数

EAC = AC + (BAC - EV) / CPI = BAC / CPI

BAC

AC

PV

PV

时间

预期完成时间

项目总结  
持续改善系统化

准备阶段  
启动阶段  
执行阶段

TSP 项目总结

准备阶段  
过程数据评审阶段  
人员角色评价阶段  
总结报告撰写阶段

范围管理  
时间管理  
成本管理  
质量管理

项目组长  
计划经理  
开发经理  
质量经理  
过程经理  
支持经理  
项目经理  
核心角色

其他跟踪  
日程计划跟踪  
承诺计划跟踪  
风险计划跟踪  
数据收集计划跟踪  
沟通计划跟踪  
原因分析  
纠偏措施定义  
纠偏措施管理

## 5 - 质量管理 — 质量策略

个人评审、团队评审、单元测试、集成测试、验收测试

管理对象 软件质量是与软件产品满足规定的和隐含的需求能力有关特征或特性的全体  
① 用户是谁  
② 用户需求优先级

基本策略 外部质量特性面向用户 内部反之

③ 面向用户的质量观

评审过程 PSP质量策略 ④ 用缺陷管理替代质量管理

⑤ 质量如何影响质量

⑥ 高质量产品也就意味着要求组成软件产品的部件无缺陷  
⑦ 各种评价方法的质量观

⑧ 缺陷的越少，消除缺陷越容易 代价增加降低

⑨ 团队代码 有缺陷测试最有效

⑩ 尽量模块内部消除缺陷 不取代系统测试

Design Review - 5 Design Inspect - 22 Code Review - 2 Code Inspect - 25 Unit Test - 32

System Test - 1405 (外部) 测试阶段比评审阶段发现缺陷多不能证明评审没用，

因为这可能因为评审不到位而不是评审没用。

★ 质量可以免费获取，获取高质量不一定高代价 & 更高质量更高成本 不矛盾

质量↑ 前期成本↑ 后期回报，缺陷少质量高效益好 造价修正成本↓

故总体成本↓ 编写者愿意接受收益，当评审和测试等前期投入在合理范围内，高质量可以

是免费的，但当前期投入过高时，更高质量带来的收益不如投入成本

评审检查表 = 加工后的对时间日志 缺陷日志 所有开发活动都要记录

缺陷：Defect 反映多处缺陷问题 Bug 反映了表现形式的问题

为什么用缺陷管理代替质量管理？ 用户对产品有完全性、性能、实用性……有很多期望，但能工作的是前提  
软件应该有缺陷。故软件产品的质量目标可以先结构化的基本质量缺陷，然后再考虑其他的质量目标

# 5 - 质量管理 指标

**Yield:** 只能预测不能度量 (因为默认缺陷是检测不完的)

**Phase Yield:** 某阶段发现缺陷个数 / 某阶段发现缺陷总数 \* 100%

**Process Yield:** 第一次编译发现缺陷数 / 第一次编译注入缺陷数

理想情况 In 和 Out 是事实  
修正情况 变更一半，然后向前面分摊推

Phase	In	Out	left	Yield		Phase	In	Out	left	Yield
				Phase	In					
DLD	8	0	8	0%	DLD	8 $\frac{10}{3}$	0	8 $\frac{10}{3}$	0	0
DLLR	0	4	4	50%	DLLR	0	4	4 $\frac{10}{3}$	$\frac{12}{34}$	~
CODING	16	0	20	0%	CODING	16 $\frac{20}{3}$	0	30	0	~
CR	0	10	10	50%	CR	0	10	20	$\frac{1}{3}$	~
UT	0	10	0	100%	UT	0	10	10	50% (估)	~

A/FR =  $\frac{\text{发现缺陷数}}{\text{代码评审 + 代码评审}} \times 100\%$   
 PSP 评估周期 / PSP 失败成本  
 编译 + 单元测试  
 评审考虑的其他因素  
 DSRPS 评审 低有风险  
 评审时间注意力  
 评审时机选择 编译之前 (pre)  
 环境评估 评审速度过大 不优先解决一些缺陷后面编译与 UT 的时间

Review 个人评审

Inspection 团组评审

阅读自己的代码 10 阅读别人代码并指出错误

**PQI: 精确可度量** 太高浪费  
 到 D4 就够了, 0~1  
 ①设计质量 设计时间 > 编码时间  
 ②设计评审质量 设计评审时间 > 设计时间  
 ③代码评审质量 代码评审时间 > 编码时间一半  
 ④代码质量 代码编写缺陷密度 < 10/1000LOC  
 ⑤程序质量 程序单元测试缺陷密度 < 5/1000LOC

五个值乘积 也是模块化规范化程度验证

发现效率:  
 编译 > 代码评审 > 设计检查 > 编码检查 > 单元测试 > 系统测试

DRL

以单元测试每小时发现缺陷数为 1, 其他阶段每小时发现缺陷数即为 DRL

# 5. 质量管理 —— 设计与质量的关系

PSP设计过程

PSP设计模板

## 质量路径

- 各种测试
- 进入测试前的产物质量提升
- 评审过程度量与稳定
- 质量意识和主人翁态度
- 个体Review的度量和稳定
- 诉诸设计
- 缺陷预防
- 用户质量观 — 其他质量属性

设计: PSP	动态信息	静态信息
① 目标位置	外部信息 反应(服务消息等)	功能(继承类结构)
③ 组件模块关系	操作 OST Operational	FST Functional 功能
⑤ 外部可见度量方法	内部信息 行为(状态机)	结构(属性、连接)
⑦ 内部运作机制	状态 SST State	LST Logical 逻辑
⑨ 内部静态逻辑		

OST: 定义测试场景和测试用例 — UML中用例图,时序图

FST: 消除二义性 用形式化描述

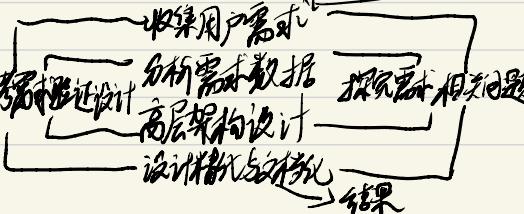
SST: 精确定义状态

LST: 消除二义性 用形式化符号描述结果 一没有对应  
UML的时序图和类图中类之间的关系,与对象交互信息,在PSP中没有

不能换顺序

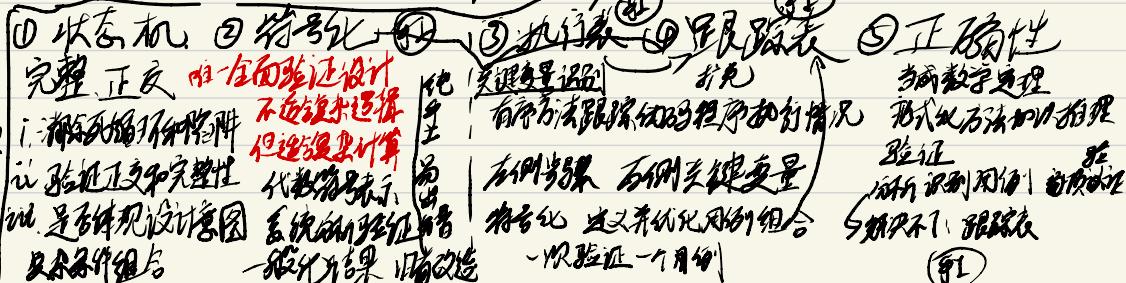
和需求开发没有关系,  
仍在用缺陷管理替代质量管理下讨论  
测试反先于评审得到改善?

设计也是排错的过程,需



系统需求 → 系统规格说明 → 高层设计  
设计说明书? 它们之间有什么关系?

对模块需求 → 代码 一样吗



# 6 - 团队工程开发

命名规范 接口标准 系统出错信息 设计类风格

团队需求开发 {  
  客户需求  
  产品需求  
  产品组件需求

客户需求靠近问题域一侧  
产品需求靠近解决域一侧

TDD: 不能提升质量, 但对软件设计显著帮助

团队设计 与PSP基本一致  
设计: ↓ 开发: ↑

团队智慧的使用 + 设计标准 + 设计复用  
+ 设计的可测试性支持 + 设计的可维护支持 → 关键功能

? 确立整体架构商分工 整合多处信息 操作概念 物品

▷ 集成策略选择: 质量优先 功能优先

大爆炸集成策略  
结合砌组件质量高

优: 质量高且效率高  
缺: 质量低时难以 Debug

逐一添加集成策略  
组件模块的本质上相同

加一个则一次  
满足仁者见仁  
测试太多成本高  
CI只是实现手段

功能测试不能自动化

集簇集成策略

系统按模块输出  
与扁平化依赖相反

提高复用性

复用接口粒度  
复用构件粒度  
生质量保证机制

扁平化集成策略

不高于  
模块化改造增加很多

尽早暴露系统级问题

但易遗漏场景  
(输入输出过滤)

低复用性

可行性:  
① 在设计阶段开始考虑  
② 为黑路径  
  考虑

验证 verification  
确认 validation

环境准备  
对象选择  
活动实施  
结果分析

验证 verification  
确认 validation

验证 verification  
确认 validation

验证 verification  
确认 validation

验证 verification  
确认 validation

Extra

## 风险管理

风险管理的目标是：在风险发生前，识别出潜在的问题，以便在完成项目的生命周期中规划和实施风险管理活动，以消除潜在问题对项目产生的负面影响。

风险管理是一个持续的前瞻的过程

任务  
边缘案例评审  
人物角色评审  
总结报告撰写

# 7- 项目支持活动

## 配置管理

建立与维护工作产品的完整性  
活动

跟踪和控制变更

建立基础 变更请求

创建基准线

变更请求小数据库

建立配置管理系统

识别配置项

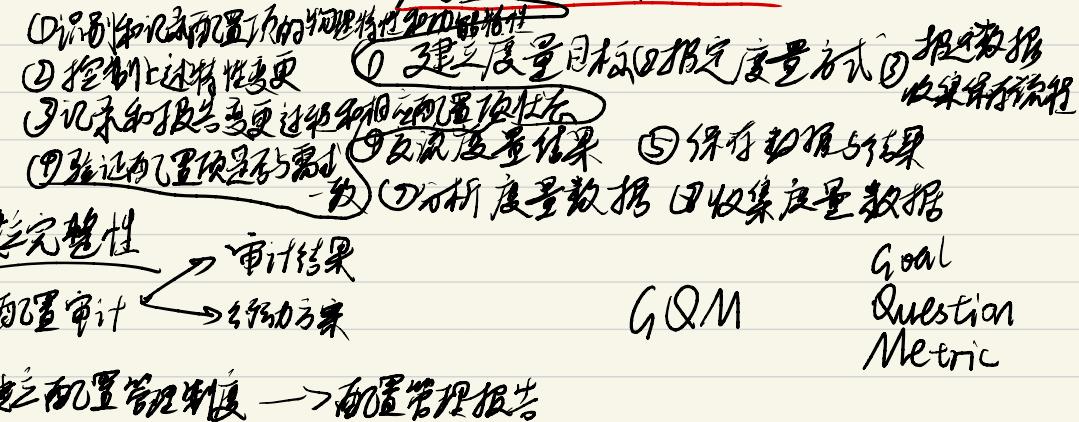
决策分析

意义与困难

- ① 建立评估备选方案的准则
- ② 识别备选解决方案
- ③ 选择评估备选方案方法
- ④ 评估
- ⑤ 从备选选择

项目招标不是决策分析活动 项目投标不是！  
最关键不是何时进行

## 度量与分析活动



①识别和选定问题

②根因分析

③行动方案

④实施改进 评估效果

## 根因分析

避免类似错误反复发生

鱼骨图

典型：技术角度、人员角度

培训角度、过程角度

根因分析有详细数据最好，用痛点、问题和缺陷的  
根因分析不一定最有正确答案

# 8- 定量管理与仿真建模

定量管理—高/低 CM/CMM 对比

对偏差的理解 Arg 9-11w E: 10w 是否操作? (数据分布)

构建定量模型 子过程能力基础 监控影响子过程关键因素  
过程模型 应用模型: 监控影响子过程的关键因素

子过程性能 遵循某个特定(子)过程之后产生结果的量化描述

时间缺陷概率 过程度量  $X_i$ : 产物度量  $Y_i$ : 缺陷密度 相应时间等

子过程性能基础 过程或子过程性能模型 性能基础和整体过程有效性的结合

选择关键子过程 → 为子过程建立度量定义 分析子过程并建立能力基础  
建立能力模型

建立项目定量管理目标 → 构建整体过程 → 选关键子过程与相应属性 (项目定量管理计划)  
监控被选定的子过程性能 → 监控项目性能 → 过程偏差根因分析 (选定度量和折衷值)

定量管理技术通常被分为非统计技术和统计技术

统计技术: 许多决策在不确定条件下做出, 故要量化不确定性 指导采取行动

非统计技术: 数据集整体特征或关联关系 助力选择适用统计技术  
检查表、柏拉图图、直方图、因果图、散点图

特殊直方图  
频率统计  
头尾风暴  
集思广益

按照高/低高/低判断过程  
已积累量折线  
是否正常  
是缺陷还是  
鱼骨图  
强/弱  
正负相关

整理 观察

收集到的数据

统计过程控制图 回归分析 方差分析 预测区间  
假设检验 敏感性分析

控制图 上限 下限 正态分布 S  
两类错误 I 假警报 II 漏警报

回归分析 回归方程 预测模型  
+ 相关性分析 (相关系数) 方程拟合

假设检验 置信水平

仿真建模 计算机仿真模型 对真实/概念  
复杂系统的抽象 显示特征特性  
系统不确定性与随机性 动态行为反馈机制

仿真范围 定义变量 过程参数 定义 I/O  
蒙特卡罗仿真 高效事件仿真 混合仿真

两级决策: ① 实时控制单个活动子流程

② 根据当前和已完成活动时序和最终结果  
进行预测

