CMMI L3 TS
技术解决过程域
Technical Solution

咨询师：冯云显

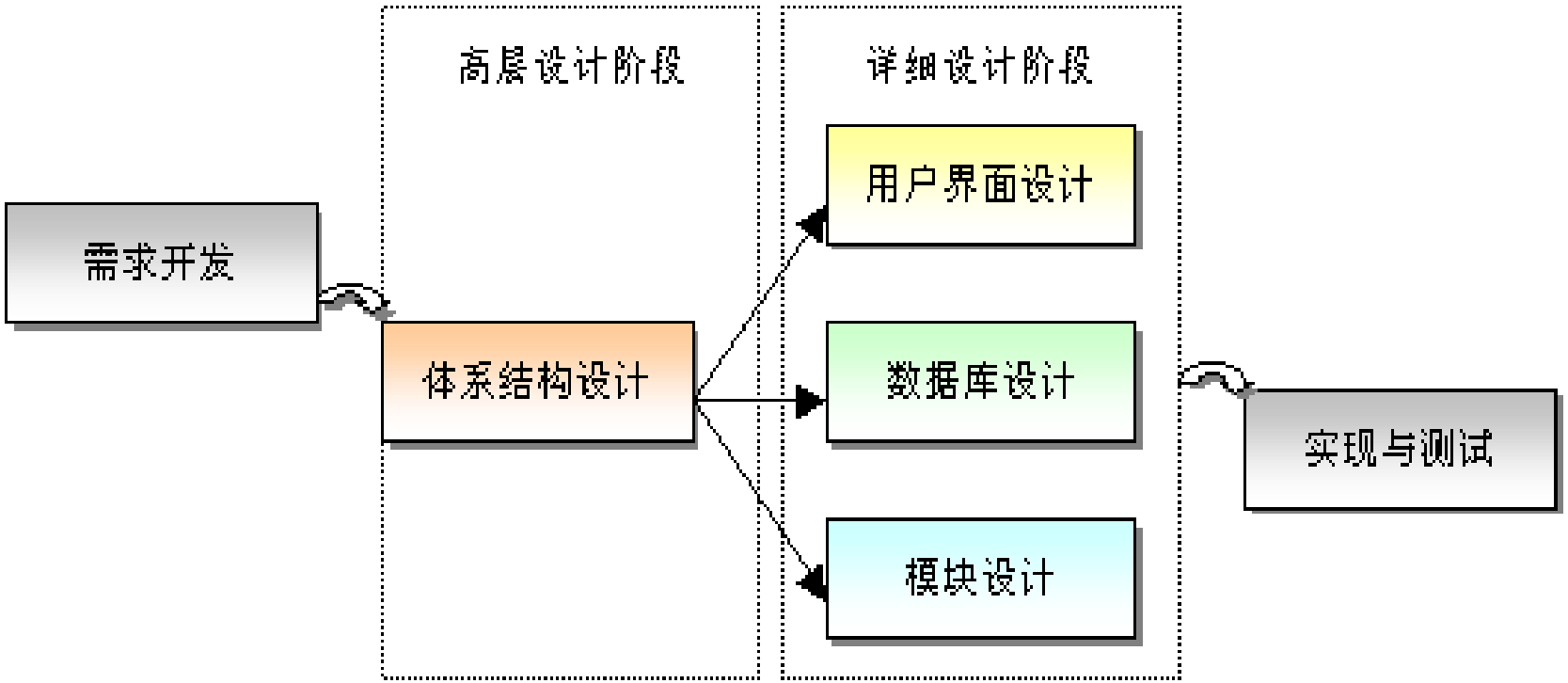
上海解元科技

主题

1、设计
2、编码
3、CMMI的TS

设计

从需求到设计

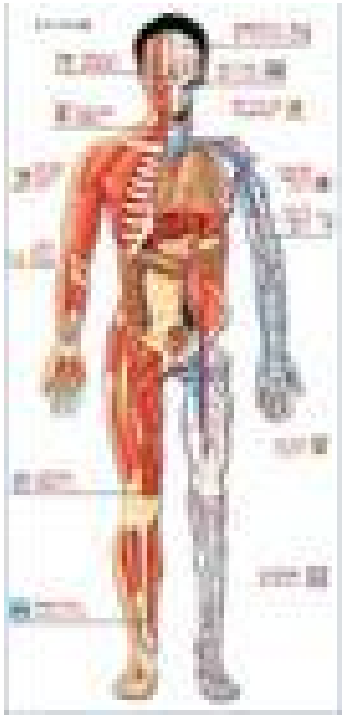


设计的层次

高阶设计阶段
- 体系结构设计

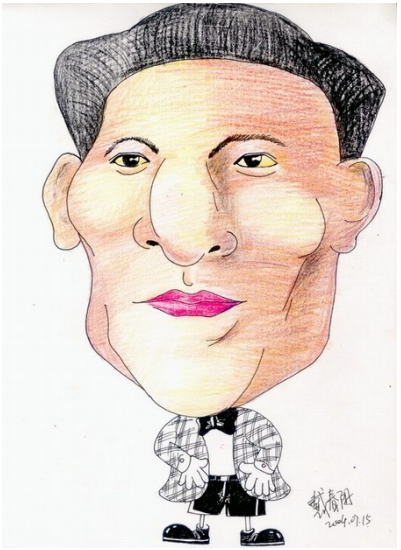
详细设计阶段
- 界面设计
- 数据库设计
- 模块设计
- 数据结构与算法

软件系统与人体的比喻



体系结构设计原则

考虑如何画一个人？



体系结构设计原则
- 合适性
- 结构稳定性
- 可扩展性
- 可复用性

体系结构设计原则-1

合适性

- 适合功能、非功能需求
- 市场决定产品命运，而不是技术
- 没有最好的，只有最合适的
- 开发多个候选方案，选择最合适的

体系结构设计原则-2

结构稳定性

- “稳定是社会发展的基础”，软件也一样
- “牵一发而动全身”，“树倒猢狲散”
- 稳定的需求决定体系结构
  易变的需求决定体系可扩展性

体系结构设计原则-3

可扩展性

- “变化”是永恒不变的主题
  是否适应变化决定了产品的前途
- 稳定与可扩展的关系：“辩证的关系”
  只是稳定，不可扩展，则产品没有前途
  稳定是可扩展的基础，结构稳定才能可持续的扩展
- 预留接口

体系结构设计原则-4

可复用性

- 可复用性是设计出来的
  - 公共部分
  - 抽象
- 复用提高生产率

CBB（公共基础构件）战略

- 项目计划时考虑CBB
- CBB是项目考核的一项重要指标
- CBB是公司业务战略的重要组成部分

Once and only once

- 相似功能的代码只能出现一次
- 切忌复制、粘贴
- 抽象

用户体验

Many good experiences···

YES
Easy to understand、
Easy to buy、
Easy to learn、
Easy to use、
Answer the customer's question at once···

I want to buy more products of this company ···

Only one bad experience···

NO
Difficult to understand、
Difficult to buy、
Need a lot of time for use、
Difficult operation、
Bad response to the question

I'll never buy a product of this company again···

用户界面设计原则

易用性
- 操作简单
- 容易使用
- 风格一致
- 及时反馈信息
- 出错处理

美观
- 布局合理
- 色彩搭配适当

以用户为中心的设计（UCD）
- 界面Demo
- 评审\体验
- 反馈
- 细化

数据库设计原则

选择合适的数据库

- 大型数据库：**Oracle、DB2、Informix、Sybase**
- 中型数据库：**Microsoft SQL Server**

根据应用软件的复杂程度选择合适的数据库

数据库优化

- 优化表结构

  规范化（第三范式）

- 优化数据库环境参数

  增强硬件性能，内存、**CPU**等

模块设计原则

透明

- 接口公开，内部实现隐藏

高内聚

- 一个模块内部各元素之间的关联度要高

低耦合

- 模块之间的依赖程度要低

模块设计的核心

- 接口（公开）
- 数据结构与算法（内部隐藏）

模块的性能决定了系统的性能

- 单个模块性能决定了系统某个功能性能
- 模块间的协作性能，决定了系统的整体性能

数据结构和算法

数据结构

- 指针
- 数组
- 链表
- 树\堆

...

算法

数学算法

...

好的数据结构和算法可以有效提高系统的性能

- 空间
- 时间

编码

- 编码规范
- 单元测试
- 代码评审

编码规范

基本的编码规范
- 排版
- 注释
- 命名

高级的编码规范
- 针对特定语言的经验总结
  - **c\c++**
  - **java**

我们是正规军还是游击队？
- 考察一个组织的开发水平首先看是否遵循编码规范

单元测试

赛门铁克误杀事件

事件经过：

2007年5月18日，在赛门铁克SAV 2007-5-17 Rev 18版本的病毒定义码中，将Windows XP操作系统的netapi32.dll文件和lsasrc.dll文件判定为Backdoor.Haxdoor病毒，并进行隔离，导致重启电脑后无法进入系统，以至连安全模式也无法进入，并出现蓝屏、重启等现象。

误杀原因：

赛门铁克在自动化系统中进行了一处改动，无意中引发了其使用的一个简单定义发生变化，导致2个Windows系统文件被错误地检测为恶意软件。

误杀事件深层分析

赛门铁克误杀事件彰显测试价值的回归

- 修改系统后，没有做回归测试，导致引发的另一个错误没有被发现
- 同时也向广大的程序员们敲响了警钟，不做单元测试的程序员在未来发展中绝对无路可走

当软件发生变化时，都要做回归测试（运行所有相关的测试用例），确保修改不会引发新的错误

单元测试价值何在？

- 增加我们对程序的信心
  测试可以让我们相信程序做了我们期望它做的事情
- 尽早发现**BUG**
  一个**bug**被隐藏的时间越长，修复这个**bug**的代价就越大
- 做回归测试的根本

程序员为何不做单元测试？

Why?

原因1

编写单元测试，增加了工作负担，会延缓项目进度?

- 编写正确的代码到底包含了哪些时间？
  编代码、调试、改错

	编码阶段发现BUG数	联调、系统测试阶段发现BUG数
做单元测试	**80%**	**20%**
不做单元测试	**20%**	**80%**

#做单元测试与不做单元测试比较

	做单元测试	不做单元测试
写代码、测试用例	写代码的同时，写测试用例，测试用例运行通过即OK	写完代码编译调试通过即OK
测试覆盖度	考虑测试的完整性，路径覆盖、分支覆盖、正常事件流、异常事件流	通常只考虑正常事件流
发现BUG数量	能够及时发现大部分BUG	通过调试只能发现部分BUG，很多BUG隐藏在代码中
是否容易定位BUG	单元测试时即能发现BUG，非常容易定位	BUG留在联调或系统测试中发现，BUG不容易定位到代码单元
是否容易改BUG	对代码非常熟悉，容易修改	间隔时间较长，需要花时间理解代码，才能修改
能否消除低级错误	通过代码静态检查、动态测试，非常容易发现低级错误，极易修改	通过联调、系统测试也能发现部分低级BUG。产品发布后也会发现低级BUG，为了一个小的低级BUG，还要重新打包、发布，增加了很多工作量
引入新BUG	修改一个BUG可能引入新BUG，通过运行单元测试用例做回归测试，给我们足够的信心没有引入新BUG	不能确定是否引入了新BUG，全面的系统测试可能会发现新BUG，但是浪费工作量；系统测试不全面，则有可能重大错误被客户发现
修改BUG时间	趁热打铁，容易修改	需要定位、理解代码，不易修改，也容易考虑不周全，引入新BUG

原因2

单元测试工作量太大，效果却不一定很好，得不偿失

单元测试需要

- 学习测试工具
- 学习测试方法
- 写测试用例

结果

- 测不出BUG

只有两种可能

- 代码质量太好了（一般不会）
- 测试方法、测试策略不对，测试用例质量不高、覆盖率低

革命尚未成功，同志仍需努力！

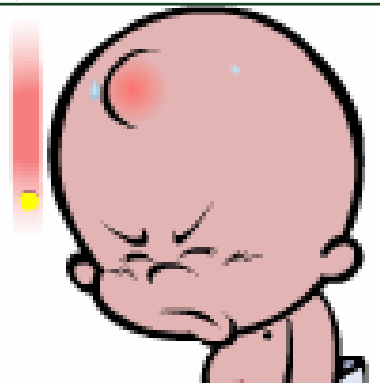
原因3

期望系统联调能发现所有BUG

如果没有单元测试，看看我们的联调是什么样的！

- 联调十有八九会失败
- 发现程序根本不能运行
- 要回过头来查找单元的错误

为什么程序不能运行？！



联调为什么不能代替单元测试？

- 覆盖率达不到要求，测试充分性不能保证
- 找错、排错工作量大
- 在想回过头来做单元测试已经来不及了

原因4

不知道怎么编写单元测试

- 利用工具

  Xunit系列（开源）、商业工具

- 测试方法

  等价类、边界值、逻辑覆盖

- 测试策略

  根据逻辑复杂度确定是否值得编写测试用例（循环、判断层次）

- 使用断言，而不是**Print**打印类

- 测试覆盖率：代码覆盖、分支覆盖、路径覆盖

原因5

项目没有要求，所以不做单元测试

项目要求什么？
- 高质量的代码

如何产生高质量的代码？
- 单元测试

你是个负责任的开发人员吗？
- 对自己的代码质量负责
- 方便别人维护、重用

你是个有潜力的开发人员吗？
- 做单元测试

单元测试是开发人员的必备技能！

原因6

我是编程高手，我不用写测试用例！

- 人都会犯错
- 系统越来越复杂
- 变更越来越频繁
- 修改BUG后，要回归测试

越是高手，越应该以身作则，带头做单元测试

原因7

越到项目后期，单元测试越难以进行

- 进度压力加大，单元测试成了牺牲品
  项目经理也不管了、开发人员也不写测试用例了
- 联调（集成）、系统测试任务加大
- BUG不断增长，项目失控

后果：

- 项目不断拖期
- 无休止的加班
- 对项目失去信心
- 大量开发人员离职
- 产品不敢推向市场

如何推广单元测试

宣传

改变项目经理、开发人员的意识

- 单元测试的好处
- 不做单元测试会怎样
- 高层出面表决心

各种方式宣传

- 海报
- 树标杆
- 讲座、论坛
- 动员会
- 其他

培训

外部专业培训
- 外来的和尚会念经

内部培训
- 切合组织实际

行政命令、制度

单元测试考核制度

- 人们只做被考核的事情
- 考核的力量是无穷的
- 公平性

考核制度是公司文化的体现

样板项目、样板个人

任何一项改革既有支持者也有反对者，我们要团结支持者，教育反对者

树立典型

- 样板项目的实际效果，用数据说话
- 样板个人的奖励措施

-榜样的力量是无穷的

-有了标杆就有了目标

专人负责推广

专人负责单元测试工作

- 懂技术
- 沟通、管理技能
- 很好的推动力

没有专人负责，推广将非常缓慢

测试工具

单元测试工具

Xuint系列(开源)

商业工具

parasoft工具集

静态代码检查工具

BUG管理工具

- Bugzilla

- Bugfree

- 其他

测试工具可以提高测试效率，专人研究工具，并在组织推广

测试用例方法

等价类划分
- 有效等价类
- 无效等价类

边界值
- 刚好等于、大于、小于边界的情况
- 经验表明，很多错误发生在边界值

内部逻辑覆盖率
- 分析代码内部逻辑，设计的测试用例满足覆盖率要求
- 可根据流程图识别

单元测试流程

简洁：可执行力强

与编码活动融为一体
- 测试先行
- 先写代码，再写测试用例
- 一边写代码，一边写测试用例

测试报告

测试**BUG**记录、分析

审计

QA审计要点

- 是否写了测试用例
- 测试用例是否达到了组织要求的覆盖率
- 是否记录了测试的**BUG**
- 是否分析了**BUG**原因，类型分布等

建立单元测试的文化

人人做单元测试
单元测试是编码的一部分
代码质量文化

- 制度保证
- 树立典型
- 强烈的质量意识

今天你做单元测试了吗？

代码评审

代码评审类型

正式审查

非正式走查

- 自查
- 交叉检查
- 四眼走查
- 会议走查

代码评审能查哪些问题

编码规范

- 排版
- 命名
- 注释

设计陷阱（特定语言的普遍性问题）

例如：C\C++语言的指针、数组等

运算符、循环、判断等常犯错误

设计、需求满足问题

- 编码是否符合设计、需求

代码评审策略

每日/每周走查

- 每天下班前交叉走查

新手必查

- 师傅带徒弟
- 有经验的高手查新手

核心代码必查

- 核心代码可采用审查方式

如何保证代码评审效果

引入代码静态检查工具

- **C++test、Lint系列（针对C、C++语言）**
- **PMD、checkstyle、Jalopy（针对Java语言）**

运用代码评审检查单

- 检查单要更新、维护
- 检查单数据要统计

检查单是很多人的编程经验、常犯错误总结，是组织的财富

编程高手

- 经验
- 培训
- 指导
- 标杆

代码评审流程

流程要求简洁高效

- 评审计划
- 合理使用评审方式
- 流程审计

编码、代码评审、单元测试迭代进行

重构

定义：不改变软件外部行为的前提下，调整程序结构，提高可理解性，可维护性

- 优化软件结构
- 帮你找**BUG**

重构时机：

- 修改**BUG**时
- 代码评审时
- 添加新功能时

单元测试用例是重构的保证

代码质量文化

开发人员日常工作

- 编码
- 代码评审
- 单元测试
- 记录缺陷
- 分析缺陷原因

质量文化

- 零缺陷
- 高质量代码是开发人员的脸面
- 奖惩制度
- **Code-Show**

CMMI中的TS

技术解决目的

设计、开发和实现满足需求的解决方案，用户文档

1 开发候选解决方案和选择准则

-根据需求分析，进行技术调研，开发多个候选方案

-根据需求分析确定选择的准则

需求中有哪些约束、限制？

-时间

-成本

-功能、性能

-其他

•候选解决方案首先在生命周期的哪个阶段来考虑的?

#候选解决方案举例

报表系统:

- 方案一:水晶报表系统
- 方案二:超级报表系统软件 等等其他报表系统与控件.
- 方案三:自己写报表控件

开发平台

- .net
- Java

2 选择产品构件解决方案

SP 1.2 选择最能满足已建立的准则的产品构件解决方案

- 用**DAR**的方法进行选择

3 设计产品或产品构件

-框架设计：建立产品的能力和产品的体系结构，包括：

- 产品的分解
- 产品构件标识
- 主要的交互构件接口
- 及外部产品接口

-详细设计：全面地定义产品构件的结构

7 设计文档

包含哪些设计文档？

- 技术方案
-概要设计\框架设计
-模块详细设计
-数据库设计
-界面设计

。。。

设计文档如何管理？

- 纳入配置管理
- 控制设计文档的变更

8 制作购买复用分析

-何时做制作、购买、复用分析？

贯穿整个设计过程，从技术方案选择开始一直到设计完成，都会考虑到制作、购买、复用分析

-采用DAR的方法进行制作、购买的抉择

-当决定购买时，要使用SAM进行管理

9 实现设计

编码

- 编码规范

  编码规范要提前培训与学习，在编码时要采用编码规范

- 代码走查

  核心代码、新手代码

- 单元测试

- 代码配置管理

  每日构建

代码验证覆盖率？

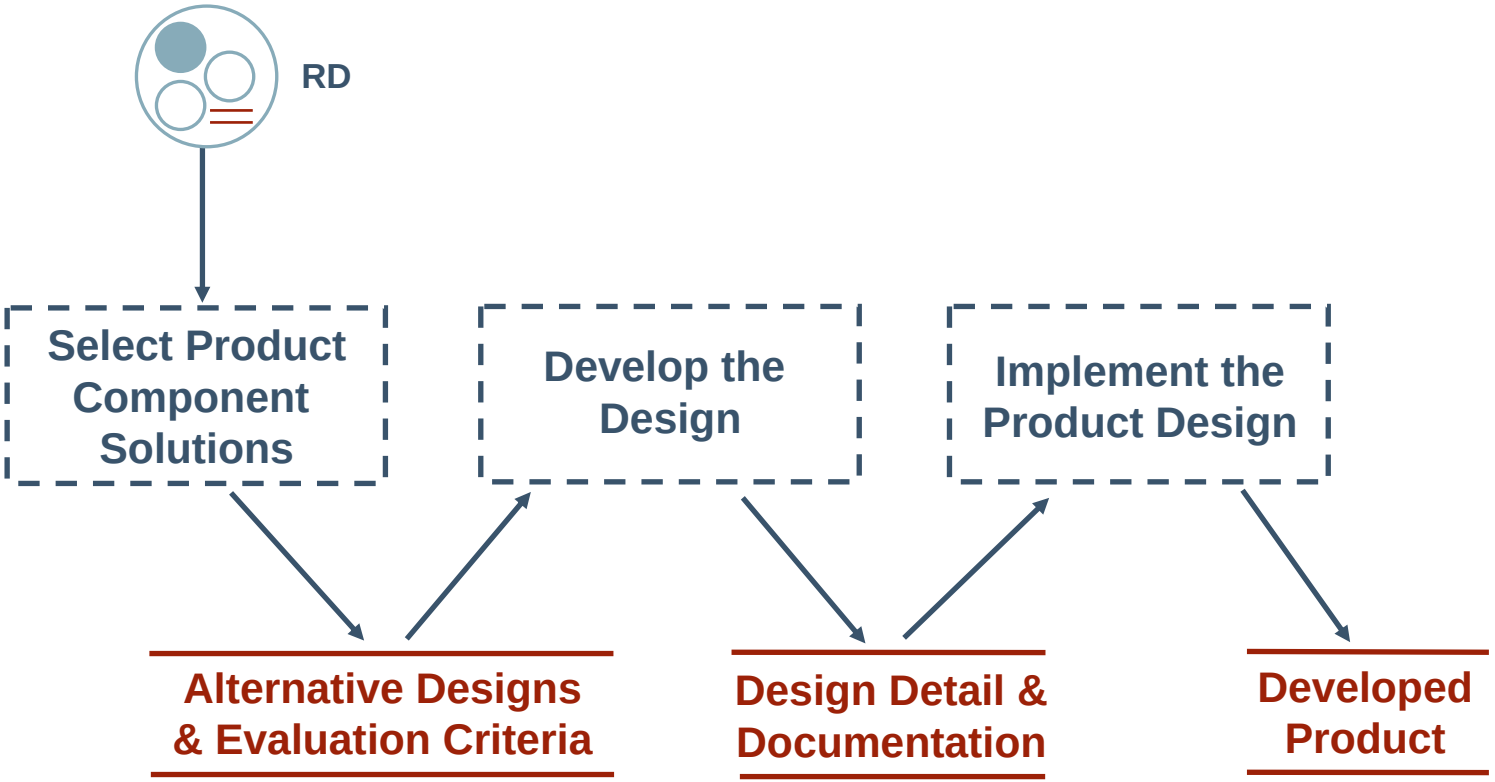
- 代码走查、单元测试综合考虑，达到代码覆盖100%

10开发产品支持文档

最终用户培训材料
用户手册
操作手册
维护手册
在线帮助

- 您认为用户文档的最佳实践有哪些?
  - 有快速入门
  - 有Q&A、FAQ
  - 有运行环境的描述
  - 保持术语一致性
  - 要有索引
  - 要有联机Help
  - 要有文档内容的链接
  - 区分读者群,不同的读者内容不同
  - 组织排版格式,字体等要求漂亮一些
  - 文档可以复用
  - 多用图表
  - 操作流程

TS语境图


RD
Select Product Component Solutions
Develop the Design
Implement the Product Design
Alternative Designs & Evaluation Criteria
Design Detail & Documentation
Developed Product

小 结 TS

特定目标	特定实践
SG1：从各种侯选解决方案中选择产品或产品构件解决方案	SP1.1 开发详细的候选解决方案和选择准则 SP1.2 选择产品构件解决方案
SG2：开发产品或产品构件设计	SP2.1 设计产品或产品构件 SP2.2 建立技术数据包 SP2.3 使用准则设计接口 SP2.4 进行制造、购买或复用分析
SG3：根据设计，实现产品构件和编制有关的支持文档	SP3.1 实现设计 SP3.2 开发产品支持文档

问题与回答