

1.264 第二讲 软件的设计步骤

案例研究 2-1 ：快速的发展

。志愿者简要地总结个案：

-1 Mickey 的观点

-2 Kim 的看法

。倘若提供动力并让程序员们自由进行会发生什么问题？

-他们是熟练的专业人士，在软件行业受到很好的训练过。

-许多的公司用动力作为首要的管理手段（例如，销售佣金，奖金）；为什么会在这里用到呢？

。用这样基本的手段，大公司是如何运行的？

。是什么原因导致设计持续很长的时间？

发展速度的多维性

。人

-最重要的事情：能力和动力

。过程

-以客户为中心

-基本原则，对程序质量的评价，风险管理，计划编制的循环周期

-“代码喜欢地狱中的人”和无序性仍然是最普通的方式。

。产品

-尺寸和特性，每一阶段的相位

。工艺技术

-产品或软件的开发环境

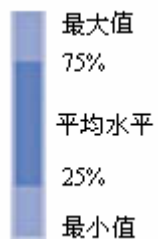
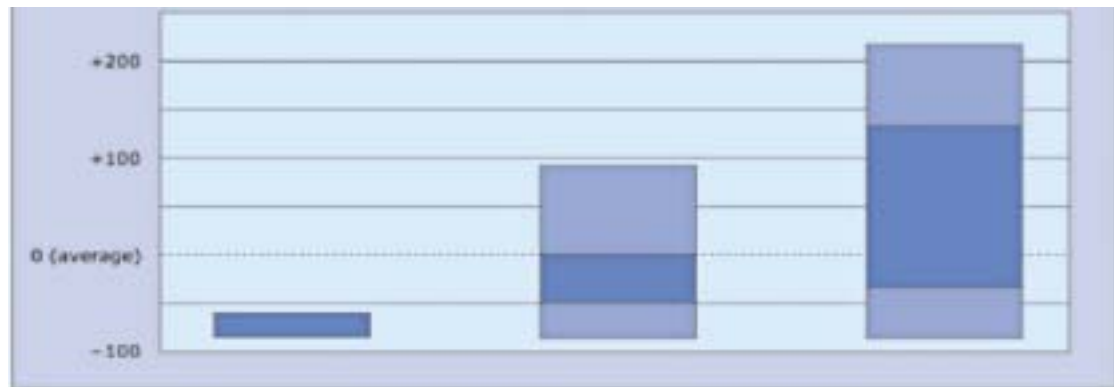
-需要使用的设备

程序设计实践

在程序的设计中，有少量测试是正确的还远远不够，不能表明程序是完全正确的，设计者应该考虑所有的情况，保证在任何状况下的正确性。

现代程序实践应用 (占总系统的百分比)

占国家生产 低 中 高
价值的百分比 (0-25%) (26-75%) (76-100%)



典型的错误

与人有关的错误	与程序有关的错误	与产品有关的错误	技术错误
装腔作势的豪言	订约的失败	特征不明显	相信新的未尝试过的技术
人员的不坚定	不充分的设计	开发者注重文采	缺乏自动源代码
痴心妄想(远离实际)	计划不充足	需求的修饰	中途变换开发工具
缺乏使用者介入	计划赶超	谈判中的拉锯战	对新工具和方法估计太高
削弱动力和积极性	对进度过度乐观	研究的导向变化发展	
吵杂拥挤的办公室	代码一团糟的编程		
缺乏购买资金保管者	在压力下放弃计划编排		
政治因素摆在首位	自下而上的欺骗行为		
设计后期加入成员	不足的控制管理		
没有重视赞助者的地位	前后模糊浪费时间		

开发者与客户发生摩擦，争执	未成熟或过度频繁的集中		
无法控制的雇员问题和管理	在评估中忽略了必须的任务		
	欺骗性的质量担保		

案例研究 2-2：快速发展

。 志愿者简要地总结案例

-Sarch 的观点

-Eddie 的看法

- 。 为什么 Sarch 采取了团队协作？为什么他们不发生口角或延迟这个方案？
- 。 他们是怎样运用工艺和技术来帮助自己的？
- 。 他们需要什么资源，多少时间来完成案例研究相对于 2-1？
- 这个差别是大还是小？会令人很惊讶吗？

案例研究 3-1：典型错误

。 志愿者概述对案例的考虑和研究

-Bill 的见解

-Mike 的意见

- 。 典型错误是怎样造成的？
- 在提到每一个原因的时候，倘若您曾经看到过或曾经卷入过，请举起您的手。
- 。 这种得处理方法在一些别的个案中也会有效吗/
- 在卖主之外发生了什么？

案例研究成果 4-1：基本原则

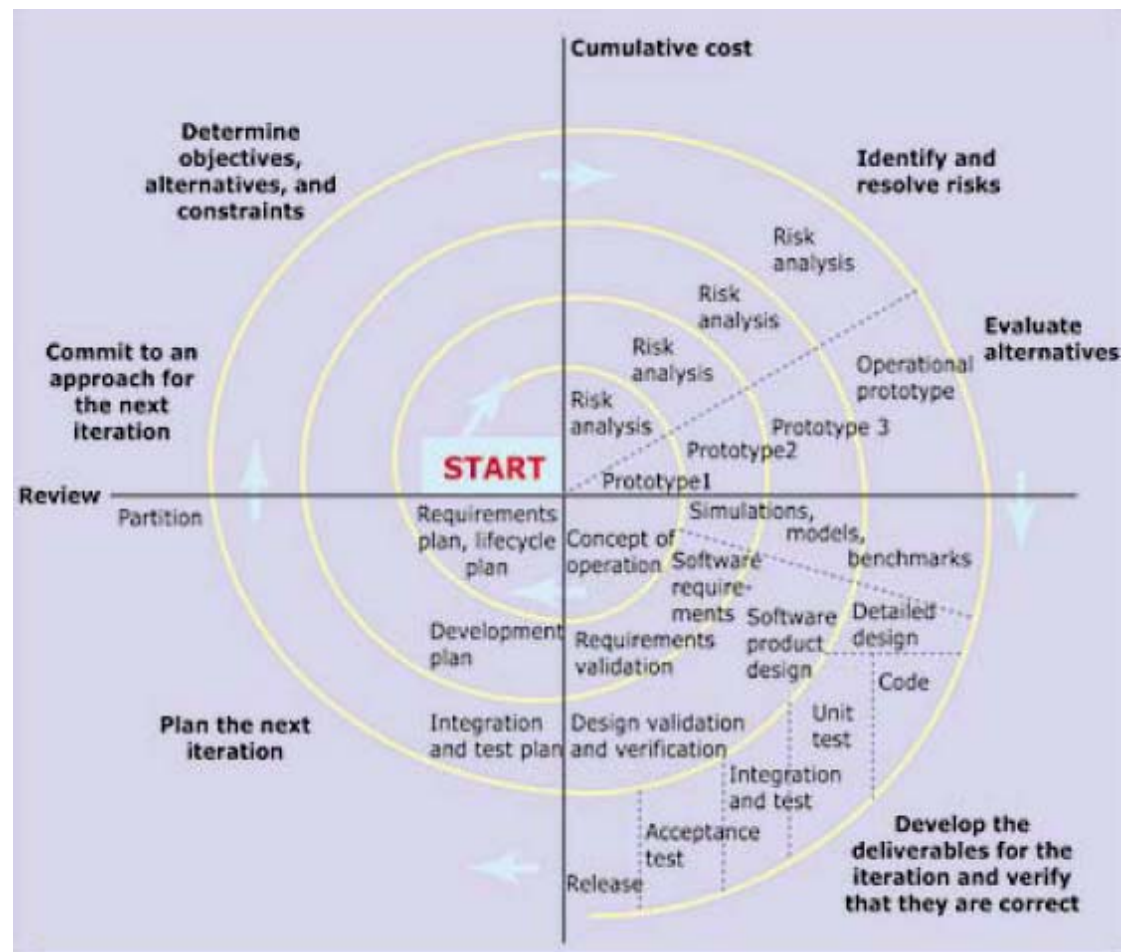
。 志愿者总结案例分析研究

-Bill 的看法

- 。 如何要避免这些错误再次？
- 最重要的一件事是什么？
- 下一个最重要的事？
- 。 你相信下一个方案回成功，在这家公司？为什么，为什么不？

技术上的基本原理

螺旋式的模式是以发展为基础的。



需求的基本原理

- 。需求：系统是干什么的/
 - 我们这学期所做的家庭作业本质上是需求分析的扩展。（见第一个螺旋）
 - 作业 2 是需求分析的一个非常的预备阶段。
- 。分析使用 UML (UNIFIED MODELING LANGUAGE) 统一建模语言。
 - 数据建模(实体的关系图表,下一个星期再讲)严格的说,不是 UML 语言的一部分。)
- 。分类图表
 - 静态但是完全的关系图表是在所有的对象之中的。
- 。顺序和协作模型（资料图表）
 - 在系统中，对成倍增长的数据流的查勘和控制
- 。状态模式（状态转换表）
 - 对数值和逻辑值动态和完全的查看。
- 。数据字典

设计的基本原则

- 。设计：系统是如何工作的？
 - 对象和数据库的设计（下个星期再讲）
 - 系统构造（在期末讲）
- 。子系统，分层/库，通信
 - 系统设计原则
- 。国际化，轻便，性能，记忆力，有效性，可靠性。。。 (也叫作非函数需求)
- 工具，设备选择：数据库，中间设备，应用设备，检测跟踪。。。。。。

发展的基本原则

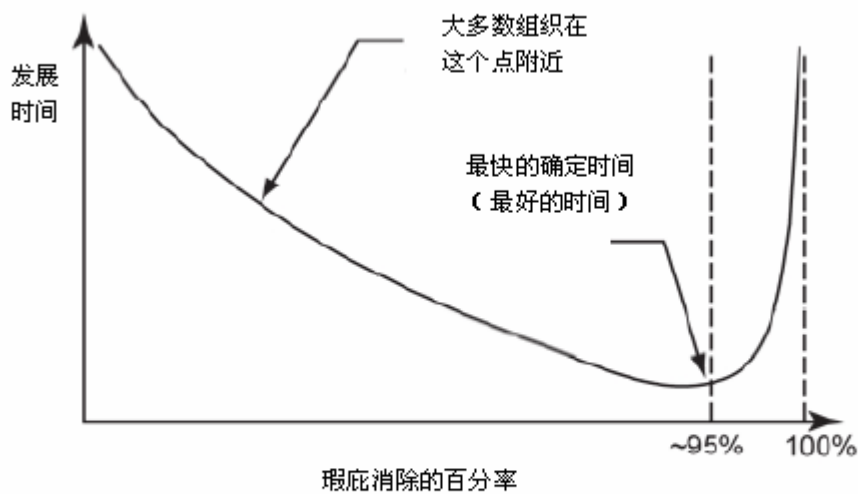
需求和设计指示着发展的成败

- 60%的过失存在在设计时间里。
- 。编写代码练习和调试练习
- 。评估练习：需求，设计，编码回顾
- 。综合策略：增长
- 。代码调整和执行
- 。CASE (Computer aided software engineering) 计算机辅助软件工程方法
 - 软件构造管理 (可靠的视觉来源，其他)
 - UML (统一建模语言)，数据模式

质量担保原则

- 。测试
 - 单位测试会发现 10—50%的缺点
 - 系统测试会发现 20—60%的缺点
 - 回顾和检查会发现 60—90%的缺点：比测试更具批评
- 。错误倾向与模块：识别和重写
 - 57%的错误存在与只占 7%的模块里 (IBM)
 - 如果平均发展速度 1000 行代码有碍 10 处错误，重写
- 。在方案的开始就要开始质量测评
 - 需求的整理和回顾
 - 设计回顾
 - 代码检查：检查员，作者，抄写员

过失预防和清出



由美国麻省理工学院
开放式课程描绘

如果在发表之后发现有 5% 的错误，那你就有麻烦了。

超级页面

超级页面 (黄色页面 (里面有很多网址))

-原形 1994 年秋天 不是个作品，很多捷径

下决心开始 1994 年 12 月

1995 年 3 月开始投放市场，我成为了经理在同年 1 月

-服务团队：

- 。原型，特别有创造性，特别固执己见
- 。对象数据库，正文取回工程，美国 NETSCAPE 商业服务器，C++
- 。产品设计太老练（高级语言 C++ 的特征），只有 2 个人会编写代码（2 个设计者）
- 。时间长，进度的压力，2 个开发个有满肚牢骚
- 。每一个开发者都没有时间写下设计和评论。

--后台服务队

- 。至少 TELCOS 知道后台服务占了一半甚至更多的分量。
- 。相关数据库，C 代码，非常的谨慎
- 。团队可以按比例增加下级开发者；实质的指导者和评估是莫认的。

超级页面

-1995 年 4 月在不是很可靠的情况下投放市场

- 。以原代码为基础：时常出错，低效的性能，(但还不错)
- 。硬件没有及时的得到安装。
- 。大量的数据问题：黄页数据不理想。

--工程重建构造

- 。让后台最好的开发者去领导服务器对；让最有经验的开发者去领导技术上的工作。
- 。把工程转移到语言学系。
- 。两个服务器开发者转移到别的工程（“工资的奴隶”，选自《神经》）
- 。用户界面设计师特别喜怒无常（心理学家说），但是实在很成熟。

--在 1996 年 7 月以前不能换总设计师。

- 。一年的四季出版了原代码为基础的新加功能。
- 。把需求程序放到因该得位置上，每季必须的计划，快速设计，代码评估。
- 。没有质量测评的队；指派下级的开发者去评估服务器和后台组。

--竞争：大的黄页（纽约电话公司：20 台个人电脑），大书（没有商业模式，好看的用户界面），其他一些地区性贝尔运营公司，公司目录

案例研究 5-1：风险管理

。志愿者总结案例研究

--Kim 的想法

- 。风险管理是管理项目不要犯典型的错误。
- 这是案例的风险管理还是典型的错误？
- 查看典型的错误清单在先前的幻灯片上。

案例研究 5-2：风险管理

。志愿者总结案例研究

-Eddie 的观点

- 。如果原来工程的全体职员仍然在岗会有多大不同？
- 。需求，设计，执行和测试程序需要使用什么？
- 为什么你认为是充分的？

案例研究 7-1：软件处理的选择

。问题：

- 原来被认为是最快的方法，没有考虑到代码的前后关系。
- 白痴的经理，傻顾问（庸俗的化合物）
- 需求的处理不在掌握中
- 。信息组代理邀请技术学院有这样理念的直接领导。

案例研究 7-2：软件处理选择

。一次好的尝试：

- 在仔细研究工程大小，财政，资源预算的情况下选择合适的范围和焦点。
- 选择螺旋的发展模式来管理风险
- 风险管理用在需求，职员安置，方法选择（安全的算法，但在危险起作用之前设计）
- 简单执行多数特性。