十个要点

~~过程模型~~

~~需求UML图~~

~~数据字典~~

~~判定表~~

~~软件测试策略~~

~~软件体系结构~~

~~基本路径测试~~

评审技术

~~挣值分析~~

~~风险管理~~

过程模型

概念

过程模型为软件工程工作提供了特定的路线，该路线图规定了所有活动的流程、动作、任务、迭代的程度、工作产品以及要完成的工作应如何组织

惯用过程模型

瀑布，增量，演化，并发

目标：使软件开发更加有序

所有的软件过程模型都支持通用框架活动，但是每一个模型都对框架活动有不同的侧重

瀑布模型

又称经典生命周期，它提出了一个传统的、顺序的软件开发方法，即从用户需求规格说明开始，顺序地通过沟通、策划、建模、构建和部署过程，最终提供完整软件和持续技术支持。

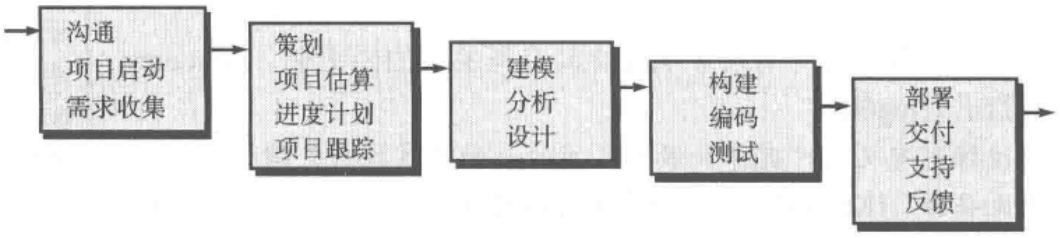


图 4-1　瀑布模型

变体：V模型

软件团队沿着V模型左侧步骤向下推进，编码结束后，团队沿着V模型右侧的步骤向上推进，其本质是增加了一系列测试（质量保证动作）

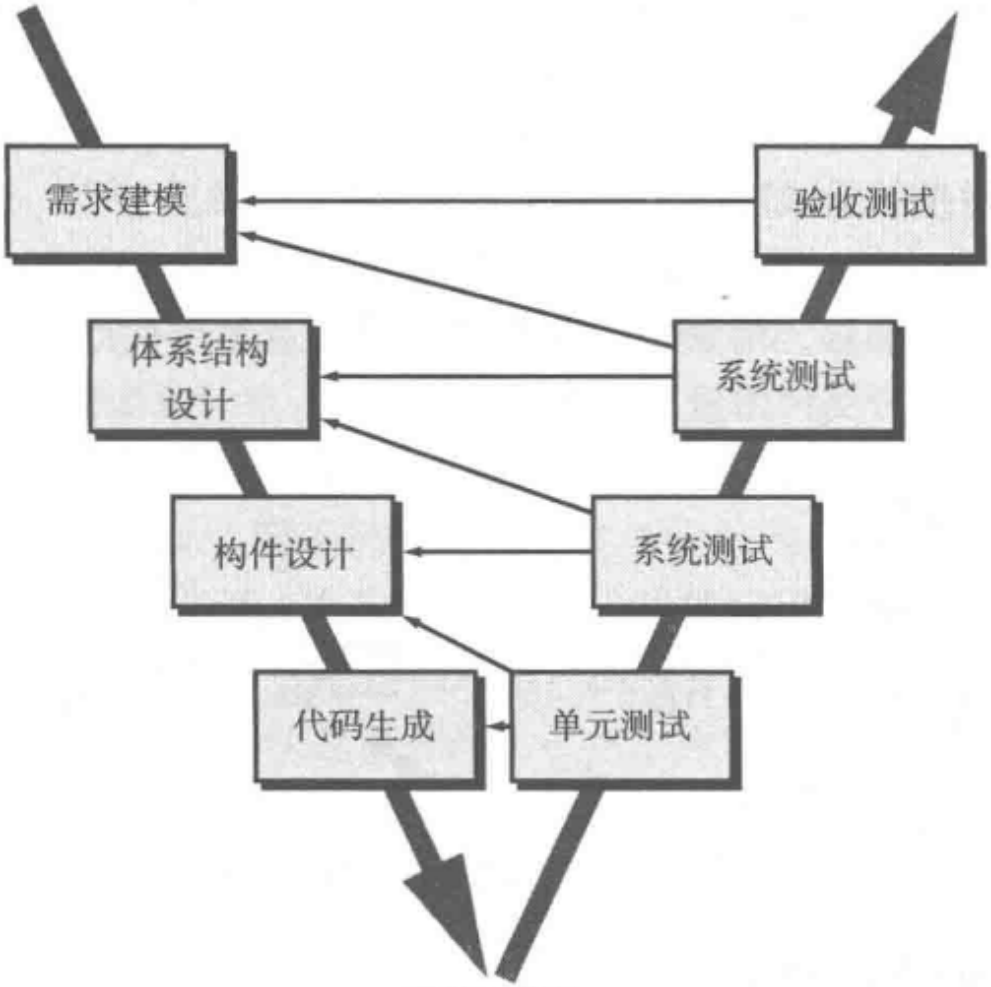


图 4-2　V 模型

对比：两者没有本质区别，V模型提供了一种将验证确认动作应用于早期软件工程工作中的方法

特点

- 阶段间具有顺序性和依赖性
  - 顺序性：只有等前一阶段的工作完成以后，后一阶段的工作才能开始；前一阶段的输出文档，就是后一阶段的输入文档。
  - 依赖性：只有前一阶段有正确的输出时，后一阶段才可能有正确的结果
- 推迟实现的观点：前面步骤完成后才考虑实现。把逻辑设计和物理设计清楚的划分开来，尽可能推迟程序的物理实现，这是瀑布型软件开发的一条重要的指导思想
- 质量保证的观点：每一阶段都需要有文档以及经过评审。
  - 为了保证质量，瀑布型软件开发在各个阶段坚持了两个重要的做法
    - 每一阶段都要完成规定的文档。没有完成文档，就认为没有完成该阶段的任务。
    - 每一阶段都要对完成的文档进行复审，以便尽早发现问题，消除隐患。

问题

- 不适应需求经常发生变更的环境
- 客户只能到项目开发的晚期才能得到程序的可运行版本，大的错误如果到这时才被发现，会造成灾难性后果
- 工作中会发生阻塞状态

所以，在需求已确定的情况下，且工作采用线性的方式完成的时候，瀑布模型是一个很有有用的过程模型

增量过程模型

提出背景

需求不明确或迫切需要为用户迅速提供一套功能有限的软件产品，然后再后续版本中再进行细化和扩展功能。

随着时间的推移，增量模型在每一个阶段都运用现性序列。每个线性序列生产出软件的可交付增量。（即以迭代方式运用瀑布模式）

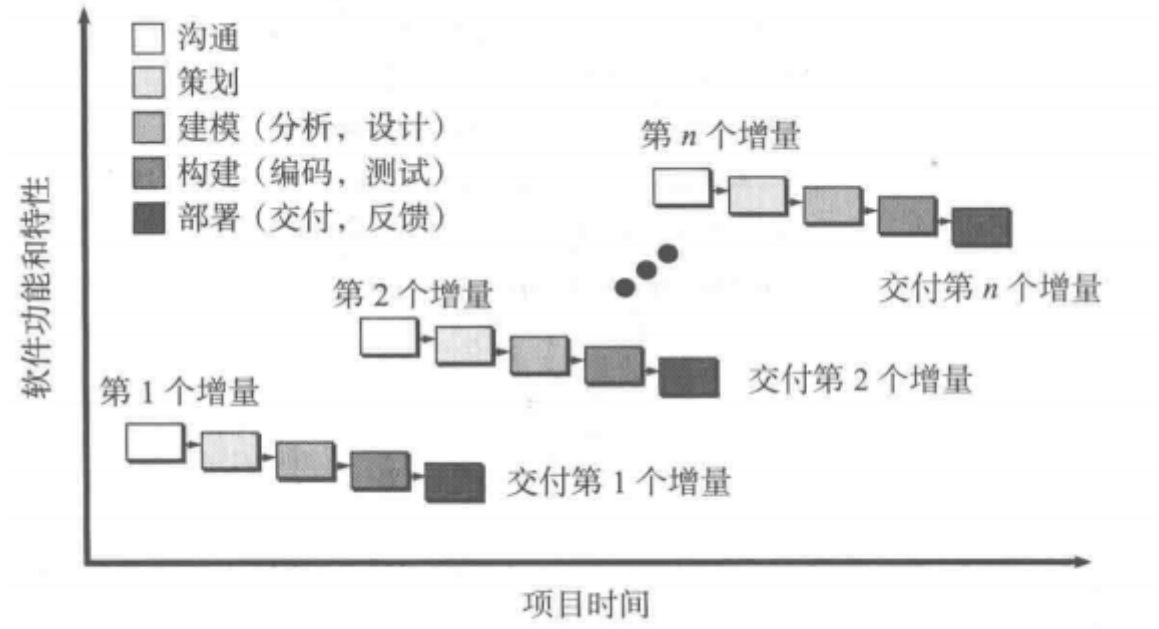


图 4-3　增量模型

运用增量模型的时候，第一个增量往往是**核心产品（core product）**。也就是，满足了基本的需求，但是许多附加的特性（一些是已知的，一些是未知的）没有提供，客户使用该核心产品或者进行仔细的评价，并**根据评价结果制定下一个增量计划。**这份计划说明了需要对核心产品进行的修改，以便更好地满足客户的要求，也说明了**需要增加的特性和功能**

【注】增量模型侧重于每个增量都提交一个可以运行的产品。早期的增量可以看做是最终产品的片段部分。但它们确实具备了用户服务能力，也为用户的评价提供了一个平台

如果在项目既定的商业期限之前不可能找到足够的开发人员，这种情况下增量模型显得特别有用

它还可以规避技术风险

优点

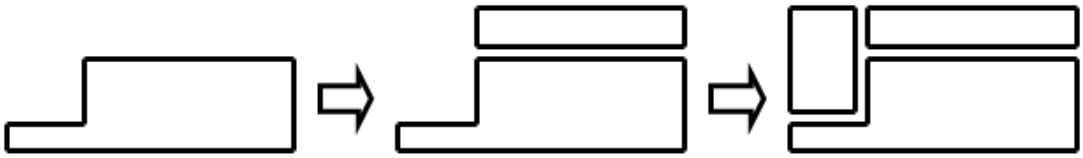
- 能在较短时间内向用户提交可完成部分工作的产品；
- 用户有较充裕的时间学习和适应新产品；
- 易于保证核心功能正确；
- 可以基于早期版本来获取需求；
- 项目完全失败的风险小；
- 可以为那些创新的功能开拓市场；
- 规避了资源缺乏的风险；

问题

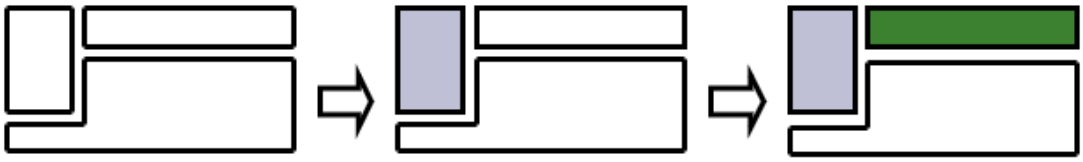
- 把用户需求转化为功能递增的不同版本可能比较难 (功能联系紧密，难以完全分开)
- 难以确定所有版本共需的公用模块。(通常进行设计时会先考虑设计公用模块，但是每一个增量只考虑局部的设计，因此，全局的公用模块很难确定)

变体——迭代开发

迭代式开发是增量式开发的一种变体，不同于传统的增量开发（每次提交一个构件），迭代式开发开始提交所有的模块（部分模块有待优化），在其后的阶段逐渐优化。



增量式开发



迭代式开发

演化过程模型

提出背景：

- 开发过程中，业务和产品需求经常变化；
- 严格的交付时间使得开发团队不可能圆满完成软件产品，但是必须交付功能有限的版本以及应对竞争或商业压力；
- 往往很好地理解了核心产品需求，但是系统扩展的细节问题却没有定义。

演化模型是迭代的过程模型，每次迭代产生软件的一个更完整的版本

常用的演化过程模型：

原型开发与螺旋模型

原型开发

背景：

- 客户提出了一些基本功能，但未详细定义输入、处理和输出需求；
- 开发人员可能对开发运行环境、算法效率、操作系统的兼容性和人机交互等情况不确定。

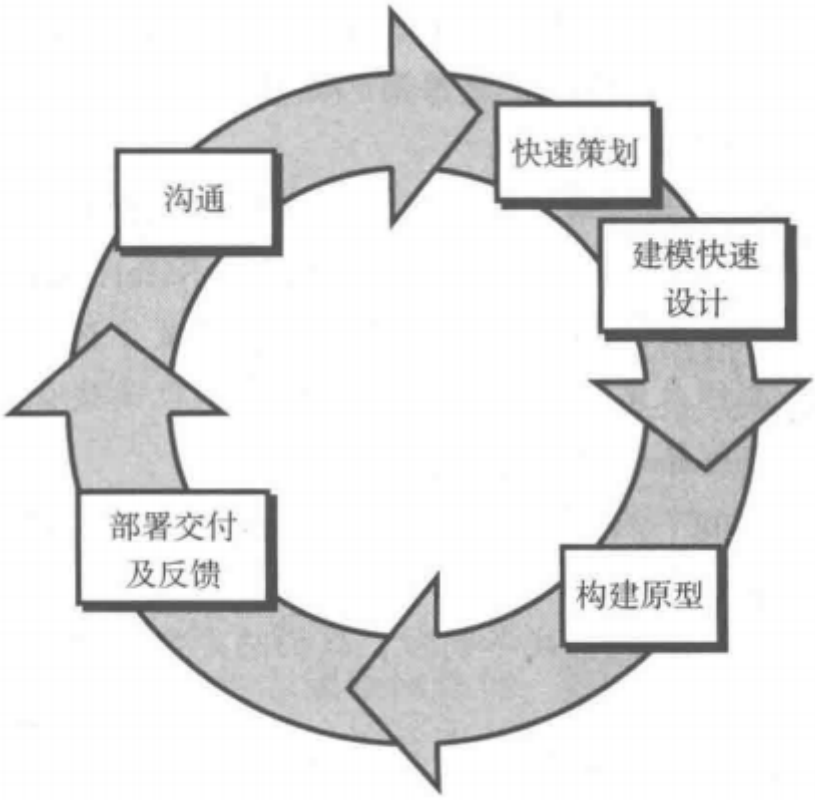


图 4-4　原型开发范型

当需求很模糊的时候，原型开发模型都能帮助软件开发人员和利益相关者更好的理解究竟需要做啥

使用前提

1. 用户必须积极参与原型的建造，建造原型仅仅是为了定义需求，之后就必须被全部抛弃(至少是部分抛弃)
2. 必须有快速开发工具可供使用

问题：

1. 利益相关者看到了软件的工作版本，却未察觉到整个软件是随意搭成的，也未察觉到为了尽快完成软件，开发者没有考虑整体软件质量和长期的可维护性。当开发者告诉客户整个系统需要重建以提高软件质量的时候，利益相关者会不愿意，并且要求对软件稍加修改使其变为一个可运行的产品。因此，软件开发管理往往陷入失效。
2. 作为一名软件工程师，软件开发人员为了使一个原型快速运行起来，往往在实现过程中采用折衷的手段。他们经常会使用不合适的操作系统或程序设计语言，仅仅因为当时可用和熟悉。他们也经常会采用一种低效的算法，仅为了证明系统的能力。时间长了，软件开发人员可能会适应这些选择，而忽略了这些选择其实并不合适的理由，结果造成并不完美的选择变成了系统的组成部分的情况。

【注】原型是为定义需求服务的，然后丢弃原型，实际的软件系统是以质量第一为目标而开发的

螺旋模型

这是一个风险驱动的软件开发模型

【注】它可以运用在应用系统开发的整个生命周期，从概念开发到维护

特点：

- 采用循环的方式，逐步加深系统定义和实现的深度（原型开发的迭代性质）；
- 确定一系列里程碑，确保利益相关者都支持系统解决方案（瀑布模型的系统性和可控性）；
- 第一圈一般开发出产品的需求规格说明，接下来开发产品的原型系统，并在每次迭代中逐步完善，开发不同的软件版本；
- 项目经理还会调整完成软件开发需要迭代的次数；

- 开发大型系统和软件的理想方法，更好地应对风险

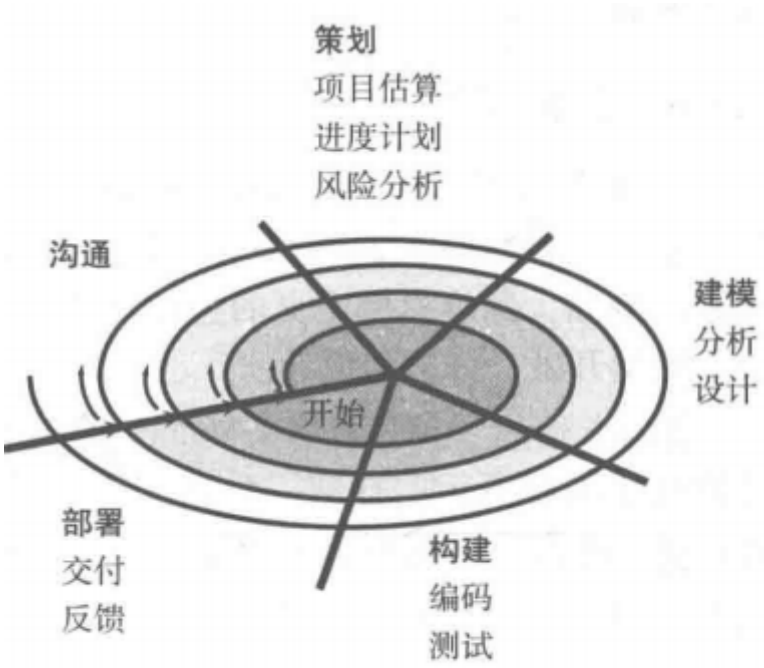


图 4-5　典型的螺旋模型

总结

演化模型的问题：

- 迭代周期数目不确定，大多项目管理和估算技术是基于活动的线性布局，所以并不完全适用于演化软件过程
- 演化模型没有确定演化的最快速度
- 演化模型侧重灵活性和可延展性，而不是高质量。（即：演化模型优先追求开发速度，而不是零缺陷）

专用过程模型

专用过程模型往往应用面较窄且较专一，只适用于某些特定的软件工程方法

基于构件的开发

基于构件的开发模型具有许多螺旋模型的特点，本质上是演化模型，需要以迭代的方式构建软件。不同之处在于，基于构建的开发模型采用预先打包的软件构建来开发应用系统

优点：能够使软件复用，减少项目开发费用，缩短开发周期

形式化方法模型

主要活动：生成计算机软件形式化的数学规格说明，使得软件开发人员可以应用严格的数学符号来说明，开发，验证系统

优点：应用数学分析的方法，歧义性问题、不完整问题、不一致问题等都能够更容易地发现和改正。在设计阶段，形式化方法是程序验证的基础，使软件开发人员能够发现和改正一些往往被忽略的问题。

缺点：

- 开发非常耗时，成本也很高；
- 只有极少数程序员具有应用形式化的背景，需要大量的培训；
- 对于技术水平不高的客户，很难用这种模型进行沟通。

应用：高度关注安全性的软件（飞行器，医疗设施，经济领域）

统一过程UP

特点：用例驱动，以架构为核心，迭代并且增量

统一过程认识到与客户沟通以及从用户的角度描述系统并保持该描述的一致性的重要性

五个阶段

- 起始阶段：识别基本的业务需求，并用用例初步描述每一类用户所需要的主要特性和功能
- 细化阶段：沟通和通用过程模型的建模活动
- 构建阶段：采用体系结构模型作为输入，开发或是获取软件构建，使得最终用户能够操作用例
- 转换阶段：通用构建活动的后期阶段以及通用部署活动的第一部分
- 生产阶段：对持续使用的软件进行监控，提供运行环境的支持，提交并评估缺陷报告和变更请求

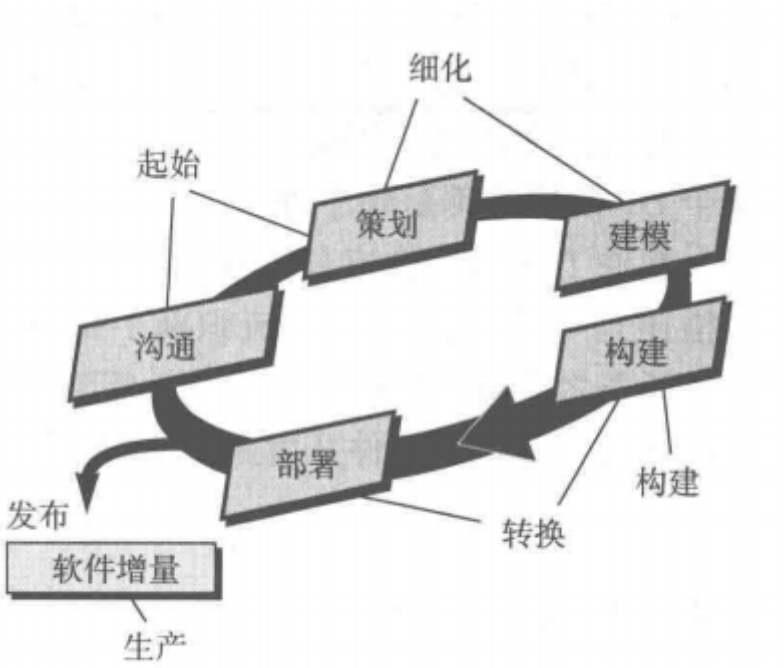


图 4-7　统一过程

数据字典与判定表

判定表

一个表格，描述条件和条件导致的动作的集合

e.g

		1	2	3	4
条件	发货单金额	＞$500	＞$500	≤$500	≤$500
	赊欠情况	＞60天	≤60天	＞60天	≤60天
操作	不发出批准书	√			
	发出批准书		√	√	√
	发出发货单		√	√	√
	发出赊欠报告			√	

数据字典

管理各种关系模型中的信息，具体信息包括：

- 一般信息：名字、别名、描述等；
- 定义信息：数据类型、长度、结构；
- 使用特点：值的范围、使用频率、使用方式；
- 控制信息：来源、使用它的程序；
- 分组信息：父结构、从属结构、物理位置等；

数据元素组成数据对象的方式：

- 顺序：两个或多个分量以确定次序进行连接；
- 选择：从两个或多个可能的元素中选取一个；
- 重复：把指定的分量重复零次或多次。

符号：

- = 等价于
- +和
- []或 (选择一个，用|隔开分量)
  例：字母或数字 = [字母字符 | 数字字符]
- { } 重复 ，(左右的数字分别为重复次数的上、下界)
  例：字母数字串 = 0{字母或数字}7 (可重复0-7次)
- ( ) 可选 (即从括号从中任选一项，也可一项都不选)

e.g

名字：订货报表

别名：订货信息

描述：每天一次送给采购员的需要订货的零件表

定义：订货报表 = 零件编号 + 零件名称 + 订货数量 + 价格 + 供应者

位置：输出到打印机

名字：试卷
别名：
描述：试卷的内容
定义：试卷 = 试卷编号 + 课程名 + 出卷人 + 时间 + 总分 + 难度
试卷编号= 1{字符}10
课程名 = 1{字符}30
出卷人 = 1{字符}8
时间 = 日期
总分 = 整数
难度 = [难|中|易]
位置：试卷数据集

软件体系结构

什么是体系结构

指的是系统的一个或者多个架构，包括软件构件、构件的外部可见属性以及他们之间的相互关系

体系结构类型

类型经常会规定特定的体系结构方法

类型隐含了在整个软件领域中的一个特定类别

体系结构风格

软件体系结构风格，每种风格描述一种系统类别，包括四个关键元素

- 一组构件，它们完成系统需要的某种功能
- 一组连接件，它们实现构件间的“通信、合作和协调”
- 约束，定义构件如何集成为一个系统
- 语义模型，使设计者能通过分析系统的组成成分的已知属性，来理解系统的整体性质

体系结构模式

风格的具体体现（体系结构设计的一个框架）

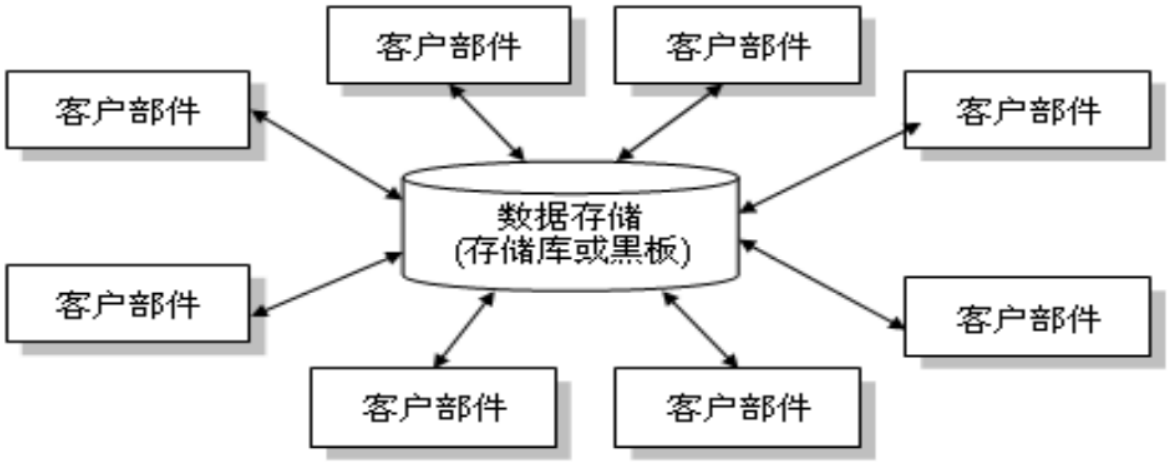
区别：

（1）体系结构模式涉及的范围要小一些，它更多集中在体系结构的某一局部而不是体系结构的整体
（2）模式在体系结构上施加规则，描述了软件是如何在基础设施层次上处理某些功能性方面的问题
（3）体系结构模式倾向于在系统结构的环境中处理特定的行为问题

体系结构风格的简单分类

以数据为中心的体系结构

数据存储位于这种体系结构的中心，其他构建会经常访问该数据存储，并进行增删改查。



特点：

- 一些数据(比如一个文件或者数据库)保存在整个结构的中心，并且被其他部件频繁地使用、添加、删除、或者修改
- 提升可集成性，即现有的构件可以被修改而且新的客户构件可以加入到系统结构中，而无需考虑其他的客户。
- 数据可以在客户间通过“黑板”机制传送，客户构件独立地执行过程。

【注】黑板：当用户感兴趣的数据发生变化时，他讲通知客户软件

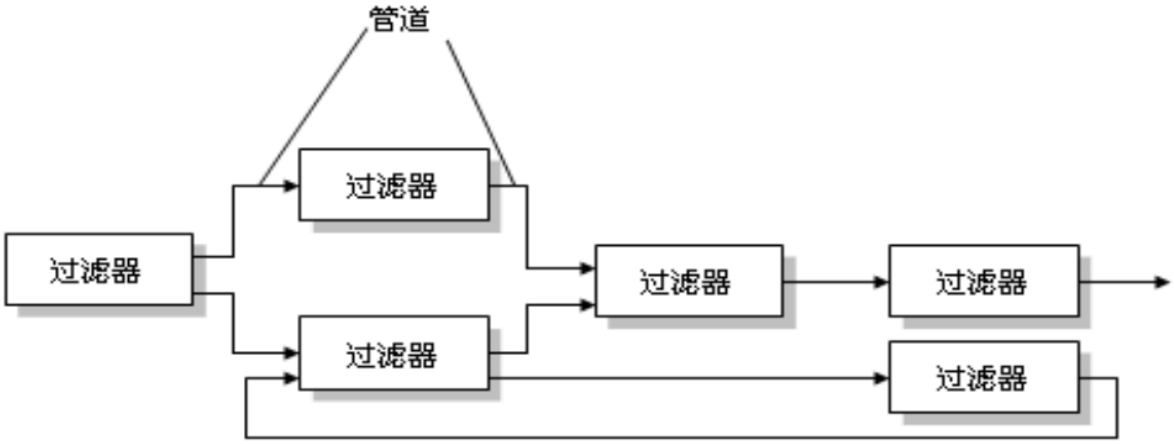
优点：

- 开放：数据对所有使用者开放
- 客户构件基本独立

问题：

- 客户软件难以协作
- 中心数据的格式必须为所有客户软件所接受

数据流体系结构



特点：

- 过滤器没有必要了解与之相邻的其他过滤器的工作
- 数据需要服从输入——变换——输出的简单流程

优点：

- 易于理解
- 过滤器易于重用
- 系统易维护
- 易并行运行

问题：

- 适用于批处理，不易于交互
- 流的协作需要考虑
- 过滤器功能重复

调用和返回体系结构

能够设计出一个相对易于修改和扩展的程序结构

存在子风格：

- 主程序/子程序体系结构

  将功能分解为一个控制层次结构，其中主程序调用一组程序构件，程序构件又去调用别的程序构件

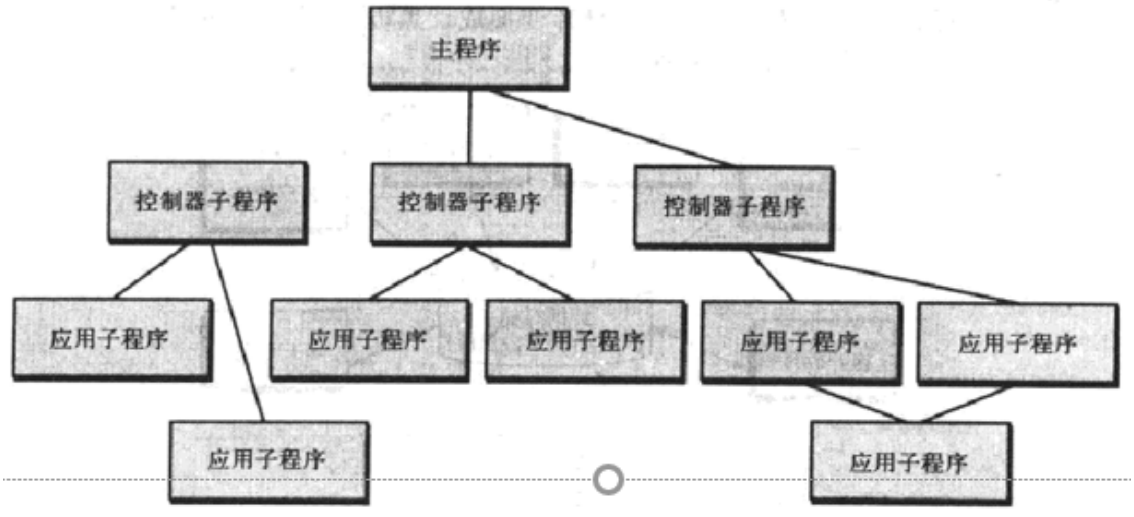


图8-3 主程序/子程序体系结构

- 远程调用体系结构

  构件分布在网络的多个计算机上

面向对象体系结构

- 系统的构件封装了数据和应用到该数据上的操作。
- 构件间通过消息传递进行通信与合作。

层次体系结构

- 定义了不同的层次，各个层次完成各自的操作
- 每一层为上层提供服务，又接受下层的服务

优点：

- 明确的抽象层次，易于增减或修改层次

问题：

- 系统并不是总能分层

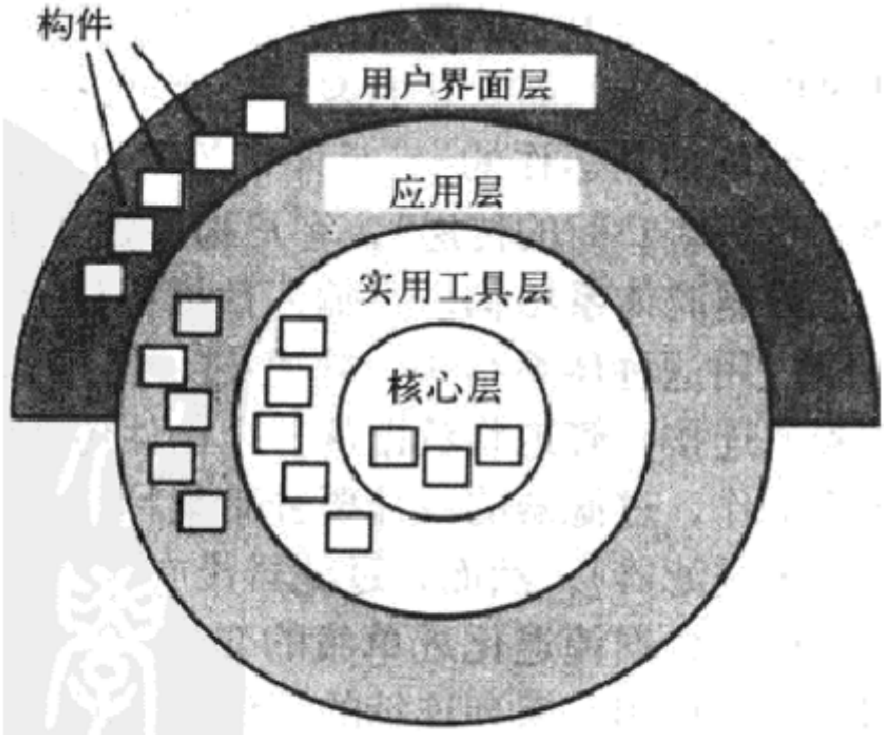


图8-4 层次体系结构

体系结构模式

体系结构模式：体系结构模式在特定坏境和一系列限制与约束下处理特定的应用问题。

模式提出了能够作为体系结构设计基础的体系结构解决方案

体系结构的考虑要素

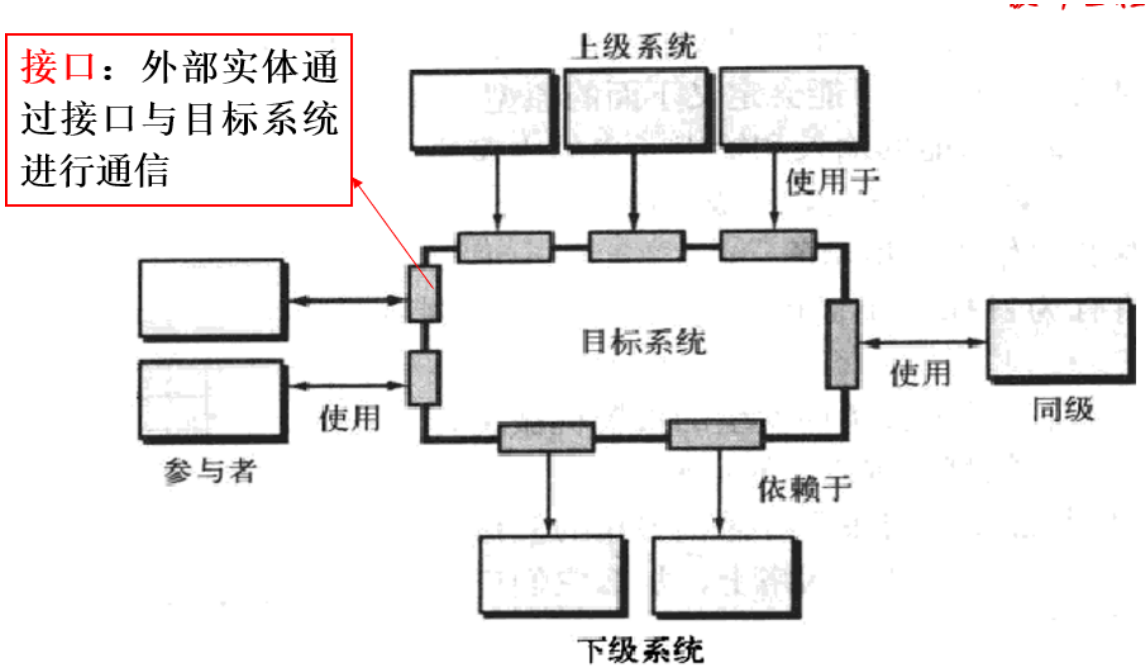
- 经济型
- 易见性
- 隔离性
- 对称性
- 应激性

体系结构设计

系统环境表示

在体系结构设计层，软件体系结构设计师用**体系结构环境图ACD**对软件与其外围实体的交互方式进行建模

ACD例子：



定义原型

将体系结构细化为构件

构件的来源：

- 应用领域：需求分析模型中的类
- 基础设施域：内存管理，通信构件，数据库构件等
- 界面领域：环境图中描述的接口隐含多个特定的构件

每一个顶层构件都必须经过反复的迭代精化

要会画UML构件图

体系结构评审

一种特定的技术性评审，提供了一种评估方法，该方法可以评估软件体系结构满足系统质量需求的能力和识别任何潜在风险的能力。

往往只设计软件工程团队成员

评审技术：

- 基于经验的推理
- 原型评估
- 情境评审
- 检查单的使用

经验学习

用决策分析和解决方案（DAR）有助于消除分歧和促成协作

- 原因链法
- 石川鱼骨法
- 思维导图

基于模式的体系结构评审（PBAR）

用于应对短暂的开发周期、紧迫的交付日期、反复变更的需求以及小规模的开发团队

体系结构一致性检查

静态体系结构一致性分析——评估已完成的软件系统是否与它的体系结构模型相符合

软件测试策略

软件测试策略包括

- 测试计划
- 测试用例设计
- 测试执行
- 测试结果数据的收集与评估

软件测试的策略性方法

测试策略必须提供

- 低级测试，来验证一小段代码是否正确实现
- 高级测试，确认系统的主要功能是否满足用户需求

测试策略为

- 专业人员提供工作指南
- 管理者提供一系列里程碑

测试进度必须是可度量的，并且使问题尽可能早的暴露

软件开发人员对负责程序各个单元的测试

在软件体系结构完成后，独立测试组ITG开始介入

软件测试策略——宏观

螺旋模型：

软件过程：沿着螺旋向内，分别为系统工程，需求，设计，编码，每走一圈都会降低软件的抽象层次

测试策略：单元测试其实与螺旋的中心，往外是集成测试，确认测试，系统测试

- 单元测试：
- 集成测试：处理并验证与程序构建相关的问题
- 确认测试：为软件满足所有的功能，行为和性能需求提供最终保证

测试完成的标准

测试永远也完不成~~~

傻了吧

策略问题

成功的测试策略

- 早在开始测试之前，就要以量化的方式规定产品需求
- 明确地陈述测试目标
- 了解软件的用户并为每类用户建立用户描述
- 制定强调“快速周期测试”的测试计划
- 建立能够测试自身的“健壮”软件
- 测试之前，利用有效的正式技术评审作为过滤器
- 实施正式技术评审以评估测试策略和测试用例本身
- 为测试过程建立一种持续的改进方法

传统软件的测试策略

单元测试

侧重于软件设计的最小单元的验证工作

对模块的五个基本特性进行评价：

- 模块接口：保证被测程序单元的信息能够正常的流入和流出
- 局部数据结构：确保临时存储的数据在算法的整个执行过程中能维持其完整性
- 边界条件：达到边界值还能正确执行
- 独立路径：确保模块的所有语句至少执行一次
- 错误处理路径

最基本的任务：选择测试的执行路径

最重要的单元测试任务之一：边界测试

模块结构测试是基础

特点：

- 侧重于软件设计的最小单元的验证工作
- 侧重于构件中的内部处理逻辑和数据结构
- 进行的越早越好

何时进行单元测试？

通常在编码完成后进行

前期应提前准备：单元测试计划，编写测试用例，单元测试代码

↑一般由白盒测试工程师，开发人员完成

依据：源程序/项目的《详细设计》文档

一般步骤：

- 编译运行，进行语法正确性验证
- 静态测试，检查代码是否符合规范
- 动态测试，深入检查代码的正确性、容错性和边界值

集成测试

集成测试指将通过测试的单元模块组装成系统或子系统，再进行测试

内容

- 单元组装后的功能正确性
- 单元之间的接口
- 集成后的系统性能

分类：

- 一步到位的集成
- 增量集成
    - 自顶向下测试
    - 自底向上测试
    - 组合方法（三明治）：用自顶向下方法测试程序结构较高层，用自底向上方法测试其从属层

一步到位的集成

缺点：

- 需要所有单元就绪，不利于开发进度
- 问题定位较为困难

但是适合规模较小的应用

增量式测试

特点：

- 集成是逐步实现的
- 逐次将未曾集成测试的模块和已经集成测试的模块结合成程序包，再将这些模块集成为较大的系统
- 不需要所有单元就绪
- 适合规模较大的应用
- 相比于非增量测试有一定优越性

自顶向下法

首先集成主控模块，然后依照控制层次向下进行集成

策略有：广度优先，深度优先

特点：

- 可能要编写很多桩程序
- 主控模块错误可能发现的比较早

主控模块用作测试驱动模块，一次用一个实际模块替换一个桩模块

自底向上法

从程序模块结构的最底层的模块开始组装和测试

特点：

- 不需要桩模块
- 要写驱动模块
- 主控模块错误发现得比较迟

如果最上两层是自顶向下集成的，可以减少驱动模块的数量（组合方法）

回归测试

在程序有修改的情况下，保证原有功能正常的一种测试策略

冒烟测试——常用的集成测试方法

冒烟测试是时间关键项目的决定性机制

活动：

- 将已经转化成代码的软件构件集成到构建中
- 设计一系列测试以暴露影响构建正确完成其功能的错误
- 每天将构建与其他构建以及整个软件产品集成起来进行冒烟测试

好处

- 降低了集成风险
- 提高最终产品的质量
- 简化错误的诊断和修正
- 易于评估进展情况

面向对象软件的测试策略

面向对象环境中的单元测试

不再孤立地对单个操作进行测试 (传统的单元测试观点)，而是将操作作为类的一部分

面向对象软件的类测试等同于传统软件的单元测试

面向对象环境中的集成测试

两种策略：

- 基于线程的测试

  对响应系统的一个输入或事件所需的一组类进行集成。每个线程单独地集成和测试。应用回归测试以确保没有产生副作用

- 基于使用的测试

  通过测试很少使用服务类的那些类（称为独立类）开始构造系统

  独立类测试完成后，利用独立类测试下一层次的类（依赖类）

  继续依赖类的测试直到完成整个系统

确认测试

始于集成测试的借书

测试集中于用户可见的动作和用户可识别的系统输出

系统测试

包含恢复测试，安全测试，压力测试，性能测试，部署测试

调试技巧

调试是使错误消除的过程，发生在测试之后

调试方法

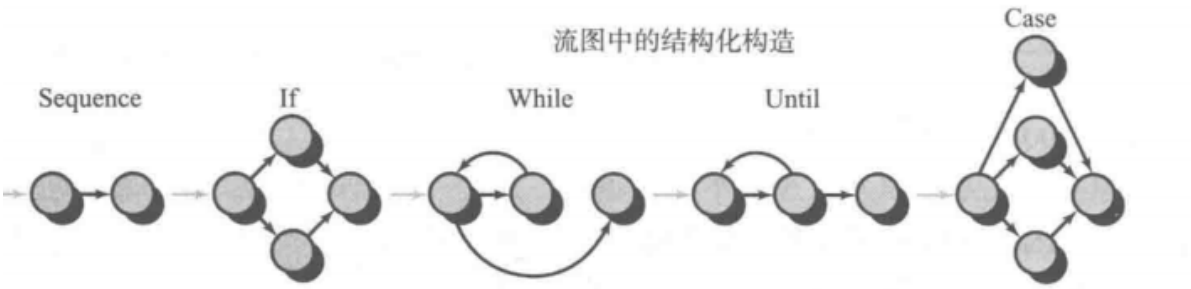
- 蛮干法
- 回溯法
- 原因排除法

基本路径测试

一种白盒测试技术

流图

- 合并顺序执行的语句
- 分离判定语句的不同条件



图中每个圆圈表示一个或多个无分支的 PDL 语句或源代码语句

图 18-1　流图表示

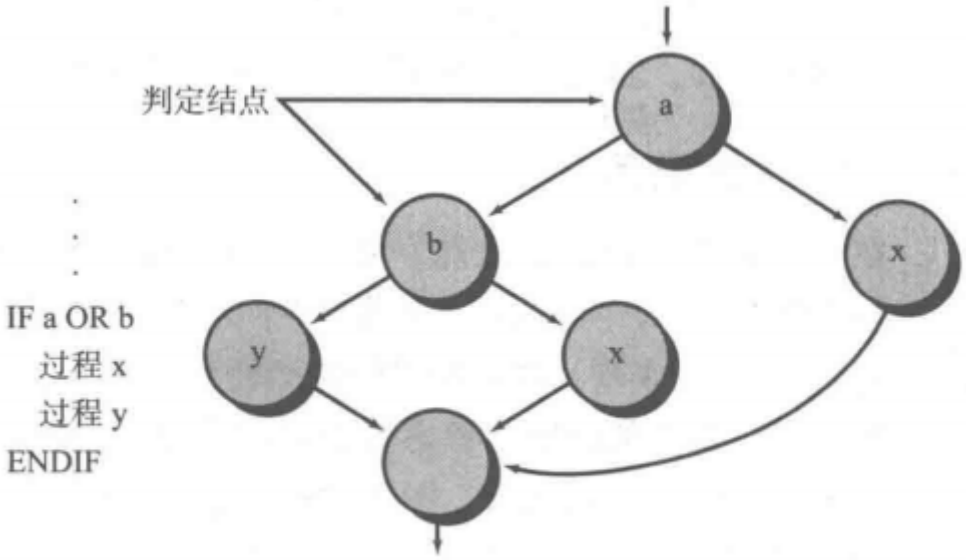


图 18-3　复合逻辑

环路复杂性

1. V(G)=E-N+2

   E是流图的边数，N是流图的节点数

2. 域的数量=环复杂性

3. V(G)=P+1

   P为包含在流图G中的判定节点数

↑V(G)的值提供了组成基本集合的独立路径的上界，并由此得出覆盖所有程序语句所需设计和运行的测试数量的上界

导出测试用例的步骤

1. 画出流图
2. 确定环复杂性，确定线性独立路径的数量
3. 确定独立路径的基本集合
4. 准备测试用例集，强制执行基本集合中的每条路径

挣值分析

按如下步骤确定挣值

1. 为进度表中的每一个工作任务确定其**预计工作的预算成本**（BCWS）
2. 将所有BCWS值加起来，可计算出**完成工作的预算**（BAC）
3. 计算**已完成工作的预算成本**（BCWP），即该时间点已经实际完成的所有工作任务的BCWS的和

   可以得到：

   进度表执行指标SPI=BCWP/BCWS（效率指标，越接近1.0则效率越高）

   进度表偏差SV=BCWP-BCWS（只表示与计划进度的偏差）

   预计完成百分比值：BCWS/BAC（应该完成的）

   特定时间实际完成工作的百分比值：BCWS/BAC(注意这里的BCWS和上面的不一样)

4. 计算出**已完成工作的实际成本ACWP**

   字如其名

   成本执行指标CPI=BCWP/ACWP（越接近1.0表示项目与预算越接近）

   成本偏差CV=BCWP-ACWP（表示在项目特定阶段的成本节省或短缺）

风险管理

风险管理策略：

- 主动风险策略
- 被动风险策略（印第安纳·琼斯）tmd，xs

工作产品：RMMM计划或者一组风险信息表单

RMMM：

- 风险缓解
- 检测
- 管理

软件风险

特性：

- 不确定性（不一定会发生）
- 损失（一旦发生就会产生恶心后果或损失）

风险类型：

- 项目风险

  威胁到项目计划。

  指预算、进度、人员、资源、利益相关者、需求等方面的潜在问题以及他们对软件项目的影响

- 技术风险

  威胁到要开发软件的质量和交付时间。

  指设计、实现、接口、验证和维护等方面潜在的问题

- 商业风险

  威胁到要开发软件的生存能力

  又分为：

  - 市场风险
  - 策略风险
  - 销售风险
  - 管理风险
  - 预算风险

另一种风险分类方式：

- 已知风险

  仔细评估项目计划、开发项目的商业及技术环境以及其他可靠的信息来源之后可以发现的那些风险

- 可预测风险

  从过去的项目经验推断来的风险

- 不可预测风险

  可能出现但是难以识别的风险

实施有效风险管理框架的七项原则

- 保持全面的观点
- 采用长远的观点
- 鼓励广泛交流
- 结合：考虑风险时必须与软件过程相结合
- 强调持续过程
- 开发共享的产品
- 鼓励协同工作

风险识别

上面的每一类风险又可以分为两种不同的类型

- 一般风险
- 产品特定的风险

识别方法

建立风险条目检查表，用于风险识别，主要用于识别下列几种类型中的一些已知/可预测风险

- 产品规模
- 商业影响
- 客户特性
- 过程定义
- 开发环境
- 开发技术
- 人员才干与经验

风险因素和驱动因子

项目管理者要识别印象软件风险因素的**风险驱动因子**

风险因素：

- 性能风险
- 成本风险
- 支持风险
- 进度风险

风险驱动因子对风险因素的影响

- 可忽略的
- 轻微的
- 严重的
- 灾难性的

风险预测

试图从两个方面来评估每一个风险

1. 风险发生的可能性或概率
2. 如果风险发生，风险相关问题产生的后果

进行以下4步风险预测活动

1. 建立一个尺度，以反映风险发生的可能性。
2. 描述风险产生的后果。
3. 估算风险对项目及产品的影响。（定性）
4. 标明风险预测的整体精确度，以免产生误解。（定量）

目的：使我们能按照**优先级**来考虑风险

> 可以把资源分配给那些具有最大影响的风险

建立风险表

e.g：

风险	风险类型	发生概率	影响值	RMMM
规模估算可能很不正确	PS	60%	2	
用户数量大大超出计划	PS	30%	3	
复用程度低于计划	PS	70%	2	
最终用户抵制该系统	BU	40%	3	
交付期限太紧	BU	50%	2	
资金将会流失	CU	40%	1	
用户将改变需求	PS	80%	2	
技术达不到预期的效果	TE	30%	1	
缺少对工具的培训	DE	80%	3	
人员缺乏经验	ST	30%	2	
人员变动比较频繁	ST	60%	2	
Σ				
Σ				
Σ				

影响值：
1—灾难的
2—严重的
3—轻微的
4—可忽略的

完成前4列内容，可以进行排序，高概率/影响的放上方，反之放下放

定义一条**中截线**，上面的风险得到进一步关注，下面的重新评估以进行第二次排序

管理角度看：

- 一个具有高影响但发生概率很低的风险因素不应该耗费太多的管理时间
- 而高影响且发生概率为中到高的风险，以及低影响且高概率的风险，应该首先列入随后的风险分析步骤中。

评估风险影响

风险显露度RE

RE=PXC

P：风险发生的概率，C：风险发生时带来的项目成本

例子：

风险识别：计划可复用的软件构件中只有70%将集成到应用系统中，其他功能必须定制开发。
风险概率：80%
风险影响：计划了60个可复用的软件构件，如果只能利用70%，则18个构件必须从头开发。

平均每个构件的程序行数是100 LOC，本地数据表明每个LOC的软件工程成本是$14.0，开发构件的总 成本将是 C = 18 * 100 * 14 = $25200
风险显露度：RE = 0.8 * $25200 = $20200

【注】

- 项目团队应该定期复查风险表，重新评估每一个风险，以确定新环境是否引起其概率和影响发生改变。
- 可能需要在风险表中添加一些新的风险，删除一些不再有影响的风险，或改变一些风险的相对位置。

风险细化

求精方法：

按照**条件-转化-结果**（CTC）格式来表示

给定<条件>，则(可能)将导致：<结果>

然后对一般条件进行求精（分析产生风险的原因）

求精出来的结论是相同点（不知道理解的对不对iai），但是求精可以帮助我们排除重大风险

风险缓解，监测和管理

风险分析活动的目的：辅助项目团队指定处理风险的策略

考虑三个问题：

1. 风险回避（通过建立一个**风险缓解**计划来实现）
2. 风险监测（监测那些可以表明风险正在变高或变低的因素）
3. 风险管理及应急计划（风险发生时的应对措施，是以缓解工作已经失败、而且风险已经发生为先决条件）

RMMM计划

RMMM——风险缓解，监测和管理计划

将所有风险分析工作文档化，项目管理者还可以将其作为整个项目计划的一部分

建立了RMMM计划，而且项目已经启动之后，风险缓解及检测步骤也就开始了。

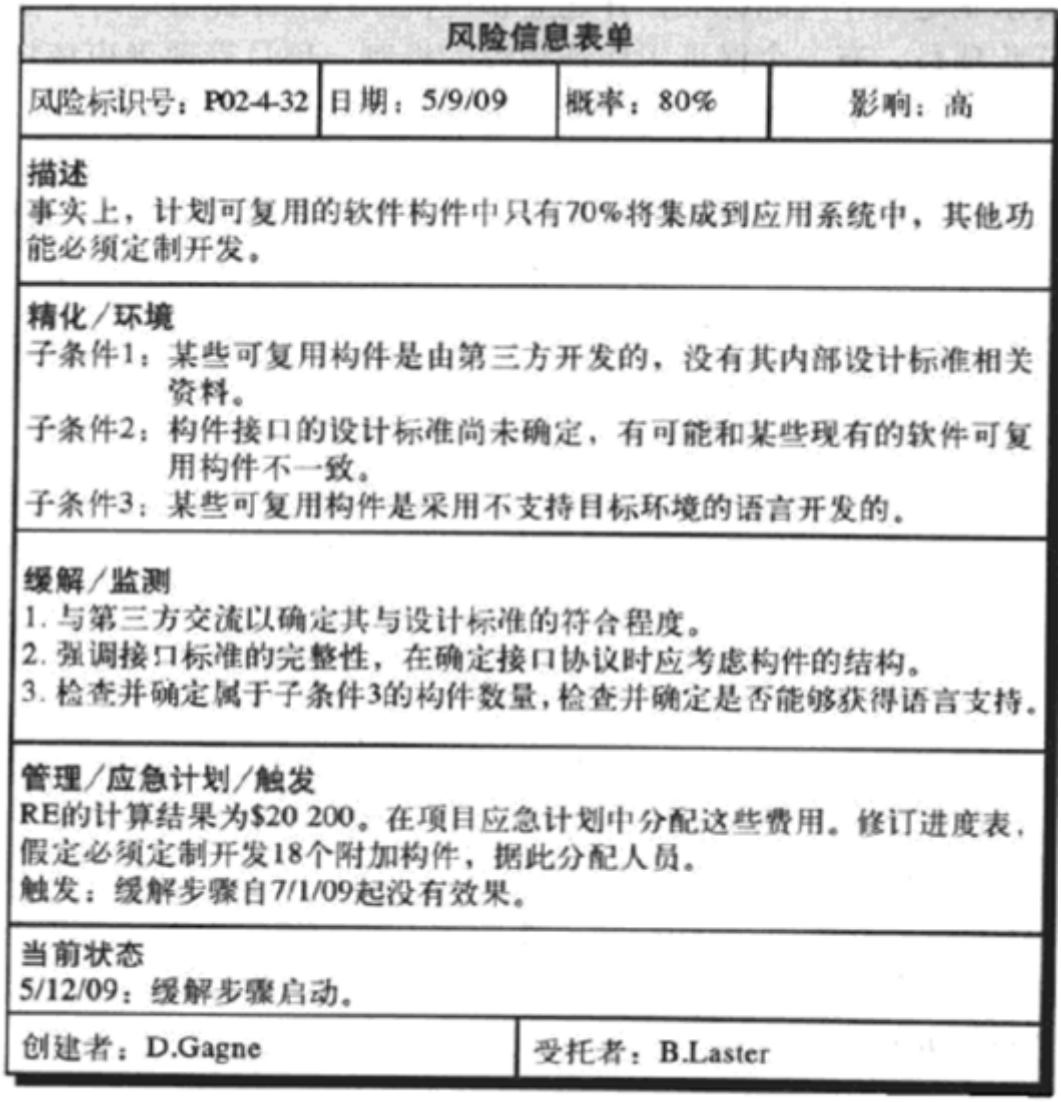
风险缓解是一种问题规避活动，而风险监测则是一种项目跟踪活动，这种监测活动有三个主要目的：

1. 评估所预测的风险是否真正发生了；
2. 保证正确地实施了各风险的缓解步骤；
3. 收集能够用于今后风险分析的信息；

在很多情况下，项目中发生的问题可以追溯到不止一个风险，所以风险监测的另一个任务就是试图找到“起源”(在整个项目中是哪些风险引起了哪些问题)。

【注】

有的团队不建立正式的RMMM文档，而是将每一个风险分别使用**风险信息表单RIS**进行文档化



风险信息表单			
风险标识号：P02-4-32	日期：5/9/09	概率：80%	影响：高
描述 事实上，计划可复用的软件构件中只有70%将集成到应用系统中，其他功能必须定制开发。			
精化/环境 子条件1：某些可复用构件是由第三方开发的，没有其内部设计标准相关资料。 子条件2：构件接口的设计标准尚未确定，有可能和某些现有的软件可复用构件不一致。 子条件3：某些可复用构件是采用不支持目标环境的语言开发的。			
缓解/监测 1. 与第三方交流以确定其与设计标准的符合程度。 2. 强调接口标准的完整性，在确定接口协议时应考虑构件的结构。 3. 检查并确定属于子条件3的构件数量，检查并确定是否能够获得语言支持。			
管理/应急计划/触发 RE的计算结果为$20 200。在项目应急计划中分配这些费用。修订进度表，假定必须定制开发18个附加构件，据此分配人员。 触发：缓解步骤自7/1/09起没有效果。			
当前状态 5/12/09：缓解步骤启动。			
创建者：D.Gagne		受托者：B.Laster	

图22-4 风险信息表单

评审技术

找不到iai，有评审的都塞进来吧

确认

对需求工程的工作产品进行质量评估

正式的技术评审是最主要的需求确认机制，确认需求的评审小组包括软件工程师，客户，用户和其他利益相关者，他们检查系统规格说明，并寻找：

- 内容/解释上的差错
- 可能需要进一步澄清的地方
- 丢失的信息
- 不一致（开发大型系统时的主要问题）
- 冲突或不现实的需求

【注】质量需求越关键，越要采用量化术语来陈述！

某些情况下，常见质量需求可以使用定性技术进行验证，在其他情况下，质量需求可以使用定性和定量相结合的评估方式进行验证

体系结构评审

一种特定的技术性评审，提供了一种评估方法，该方法可以评估软件体系结构满足系统质量需求的能力和识别任何潜在风险的能力。

往往只设计软件工程团队成员

评审技术：

- 基于经验的推理
- 原型评估
- 情境评审
- 检查单的使用

经验学习

用决策分析和解决方案（DAR）有助于消除分歧和促成协作

- 原因链法
- 石川鱼骨法
- 思维导图

基于模式的体系结构评审（PBAR）

用于应对短暂的开发周期、紧迫的交付日期、反复变更的需求以及小规模的开发团队