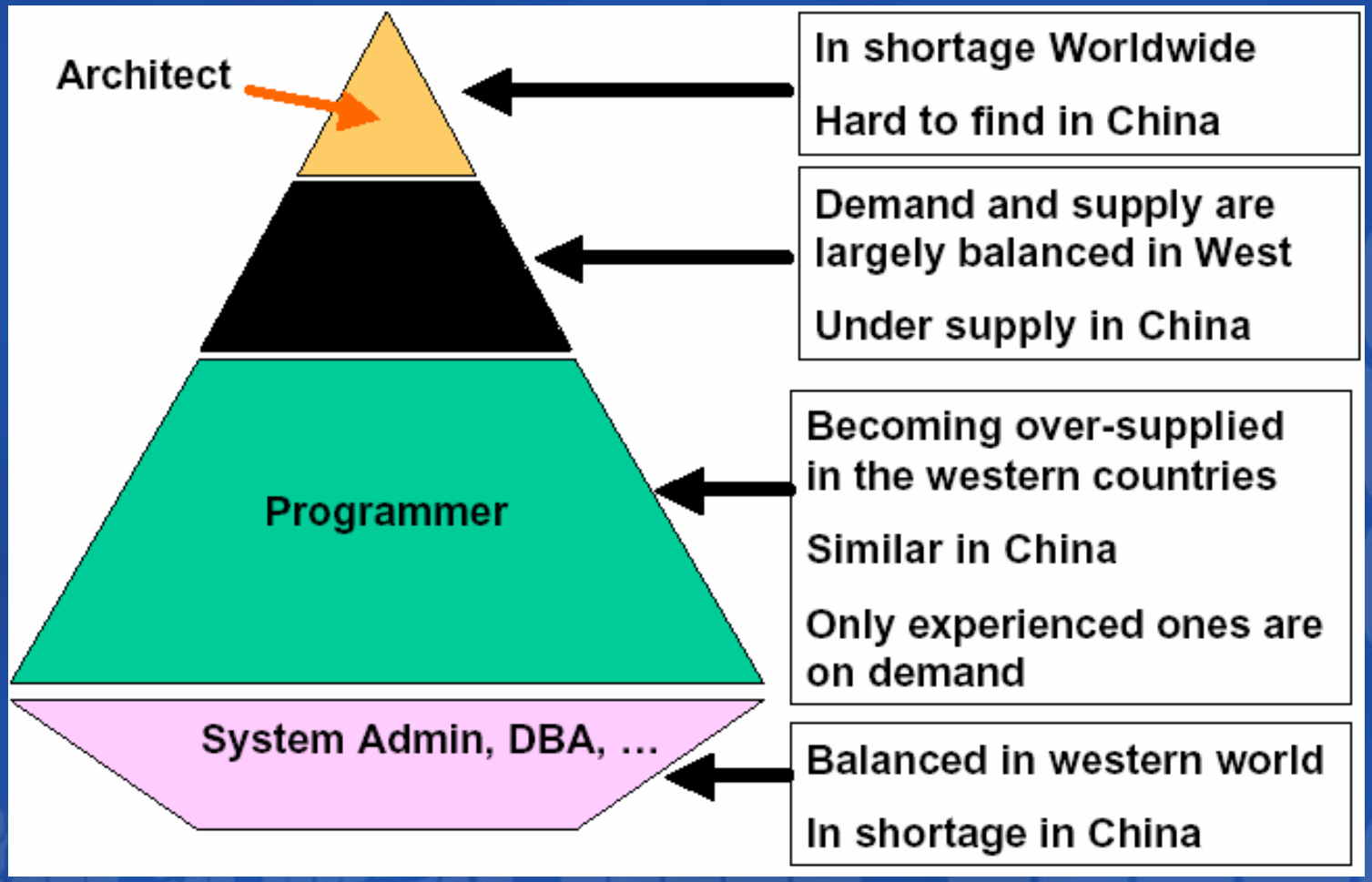
系统设计概述

Name: 汤涛
Job Title：技术总监
Company: 北京通嘉科技（www.onit.cn）
Email：tangtao@onit.cn

IT 行业的人才结构


Architect
In shortage Worldwide
Hard to find in China
Demand and supply are largely balanced in West
Under supply in China
Programmer
Becoming over-supplied in the western countries
Similar in China
Only experienced ones are on demand
System Admin, DBA, …
Balanced in western world
In shortage in China

软件架构师

- 一个奇怪的职业领域
  - 重要性日益提升、供不应求
  - 正在不断发展和成熟过程中
  - 基本上没有正式的大学课程
  - 主要由IT 行业中的专家组成
- 与建筑业的比较
  - 有专业的设计院、设计事务所
  - 有成熟、完整的规范
  - 技术已经成熟，变化与发展缓慢

软件架构设计的一些特点

- 处于软件系统建设的上游
- 需要全面考虑多方面的因素
- 对于同一个问题，可以有多种设计结果
- 在各种制约条件下取得的较好折衷方案
- 科学 + 经验 + 艺术
- “系统架构”往往被滥用

软件架构师在干什么？

- 思考、思考、再思考
  - 深入理解、准确把握建设的业务需求
  - 分析所有可见的问题、障碍、风险
  - 充分参考已有的成功方案，降低风险
- 交流、讨论、博弈、质疑
  - 对构思中的方案不断提出质疑，避免漏洞
  - 广泛听取各层面的意见，开拓思路
  - 反复质疑、逐步完善已有的设计构思
- 在动工建设之前验证设计方案的正确性

架构师的具体工作

软件架构师的知识结构

- 基础知识
  - 最好要有系统开发全过程经验
  - 对IT 建设生命周期各个环节有深入了解，包括：系统/模块逻辑设计、物理设计、代码开发、项目管理、测试、发布、运行维护、等
  - 深入掌握主流技术平台上开发系统的方法
  - 了解多种应用系统的结构
  - 了解架构设计领域的主要理论、流派、框架
- 特殊知识
  - 深入了解系统建设的业务需求
  - 了解系统的非功能需求和运行维护需求
  - 了解企业IT 公共设施、网络环境、外部系统

软件架构师的思维方式

- 基于框架的思维
  - 架构设计的层次（Enterprise, Application, etc）
  - IT 的生命周期（What, Why, Where, How, When, etc）
  - 成功经验以及方法论的指导
- 合理把握技术细节
  - 把握各个层次应有的内容
  - 合理忽略不应有的技术细节
- 风险管理意识
  - 采用成功经验、避免不应有的风险
- 多方位的开放思维
  - 多维度、多方向、包容性、避免排他性
  - 分析、质疑、抽象、归纳
  - 没有绝对好的架构设计，只有相对优秀的方案

软件架构的层次（一）

- Enterprise层
  - 说明：
    - 最高层，人数极少
  - 特征：
    - 关注整个机构、企业所有IT 系统的整体能力
    - 从整体着眼、与业务紧密相关、与IT 规划相关
- Application层
  - 说明：
    - 系统架构最高层，大型系统需要有一个架构组
  - 特征：
    - 负责应用系统的架构，奠定系统建设的基础
    - 关注系统内部的构成和子系统/模块的分划
    - 需要负责与外部相关系统的互联互通

软件架构的层次（二）

- System/Sub-System层
  - 说明：
    - 一个系统建设项目中常常有多个
  - 特征：
    - 根据应用系统的逻辑架构制定相应的技术实现方式，设计系统的物理架构
- Component层
  - 说明：
    - 常常由系统工程师担任
  - 特征：
    - 负责系统模块的实现机制和详细结构设计
    - 为系统开发建设奠定基础

软件架构的层次（三）

- Data/Information层
  - 说明：
    - 常常由数据库专家负责
  - 特征：
    - 负责应用系统的信息和数据模型和结构
    - 通常包括数据库模型和结构设计
- Security层
  - 说明：
    - 需要由安全专家负责，极缺
  - 特征：
    - 负责系统的安全架构设计
    - 涉及系统所有层面的安全措施

软件架构的层次（四）

- Network层
  - 说明：
    - 常常由网络集成商负责
  - 特征：
    - 系统内部、外部的网络拓扑设计
- Others
  - 不同建设项目常常有一些特殊需求

软件架构的分类

- 概念架构
  - 关注整个机构、企业所有IT 系统的整体能力
  - 从整体着眼、与业务紧密相关、与IT 规划相关
- 逻辑架构
  - 系统子系统、模块分划
  - 功能边界的确定
  - 分布式计算系统设计的特点
- 物理架构
  - 针对代码开发
  - 与采用的语言、技术平台紧密相关
- 数据架构
  - 数据库设计
- 部署架构
  - 对系统硬件部署
  - 与逻辑架构不同
  - 分布式系统有许多特别的性能和安全考虑

主要架构模型

- 工业模型
  - Zachman Framework
  - Meta – Enterprise Architecture
- Microsoft模型
  - Microsoft solutions framework model
  - Software factories
  - Module map & Motion Methodology

架构设计的必要性

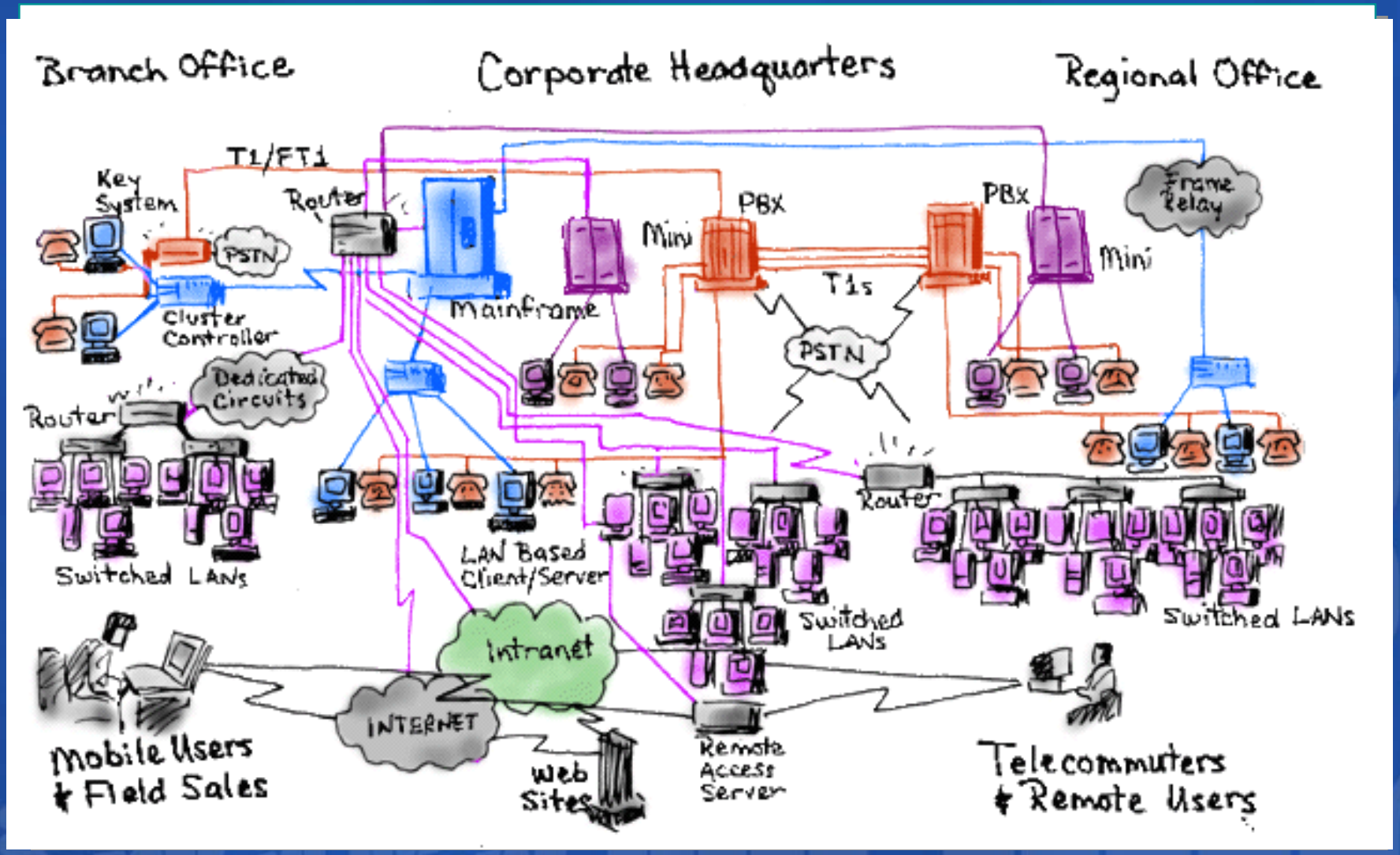
温彻斯特 “神秘”之屋

- 建设了38年– 147建筑者 0架构师
- 160个房间 – 40个卧室, 6个厨房, 2地下室, 950扇门
- 65扇门对着白墙壁, 弃用了13楼梯, 24个天窗开在地板上
- 没有任何架构图纸

软件架构设计？

错综复杂是软件的必然特征，而不是偶然属性。

我们是不是经常遇到这么复杂的架构？



为什么软件的复杂性是内在的？

- 需求本身的不稳定性
- 需要解决的问题本身的复杂性
- 管理开发过程的困难
- 需要通过软件来实现灵活性
- 描述不连贯系统的特征

. . .

系统越复杂，失败的可能性就越大！

So, Keep it simple!

软件的最佳实践

一套组织良好的文档化的规则、方法和过程，可以缩短软件开发的上市时间、提高质量、降低成本、提高工期的可预测性以及客户满意度。

- 管理需求
- 迭代开发
- 可视化模型
- 过程管理
- 组件化架构
- 验证质量
- 控制变更
- …

需求第一

需求过程

用户输入

业务问题分析 -> 用户需求获取 ->

确定产品特性 -> 编写项目前景文档 ->

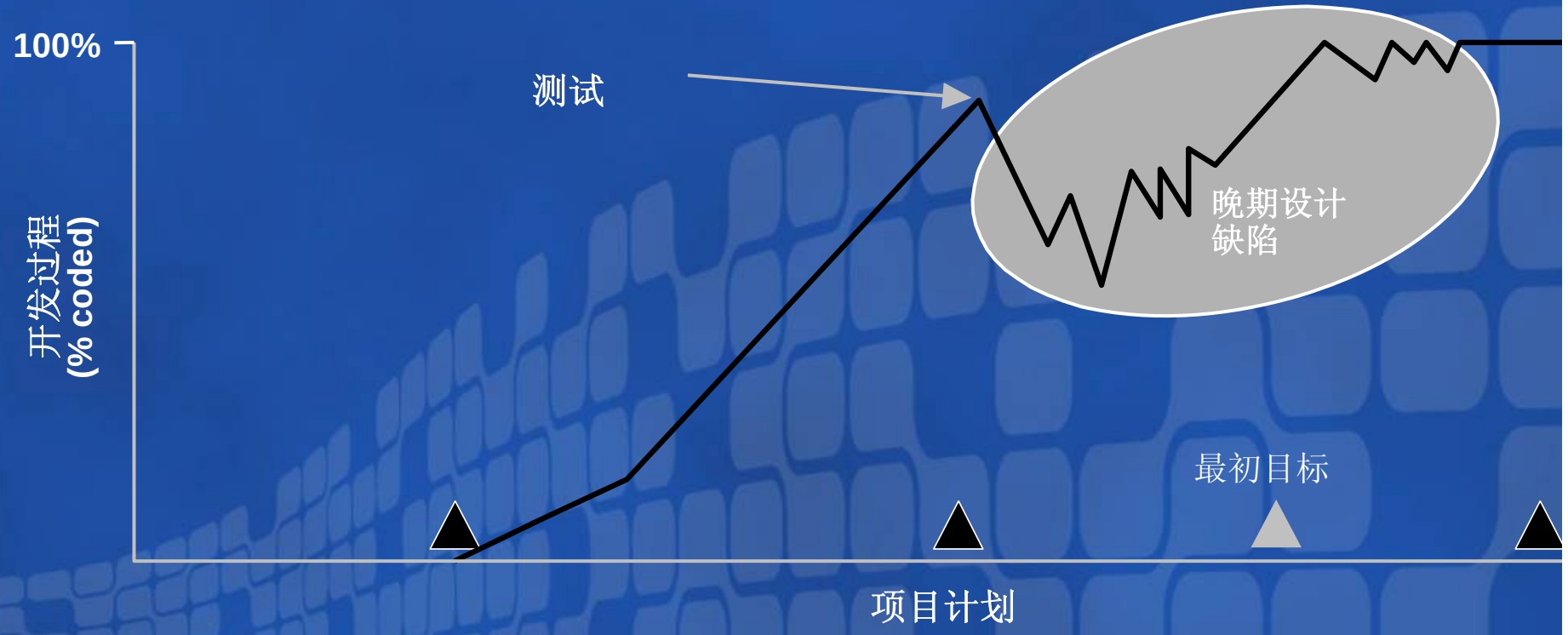
评审项目前景文档 -> 软件需求分析 ->

编写SRS -> 需求验证

功能基线

过程顺序

概念 -> 分析 -> 设计 -> 编码 -> 测试 -> 维护

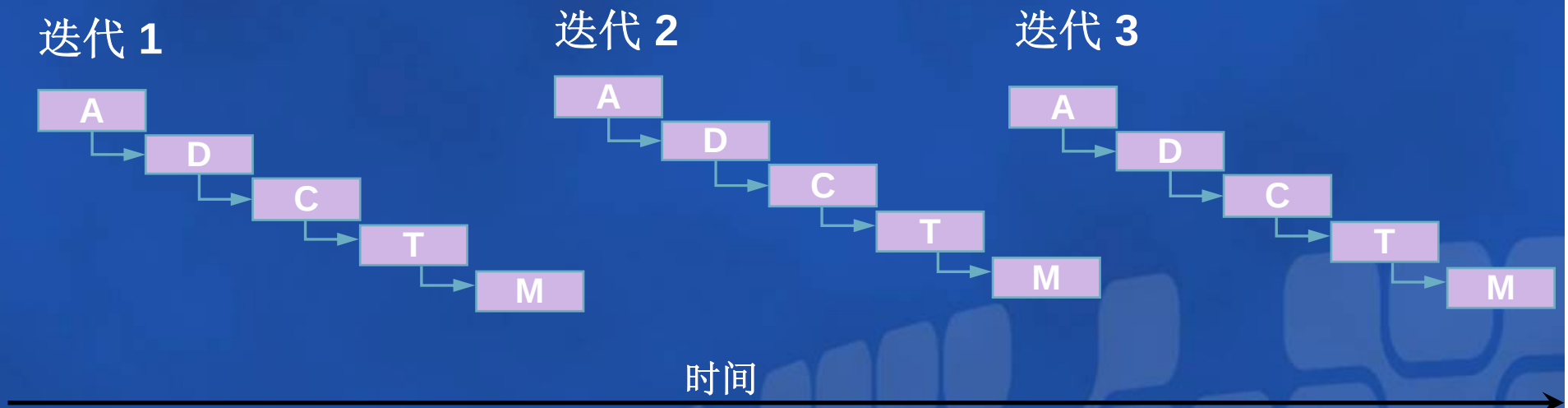


迭代化软件开发

- 针对关键需求的最初设计很可能是有缺陷。
- 后期阶段所发现的缺陷将导致项目成本剧增或者是取消项目。

为了实现缺陷设计所耗费的
时间和金钱是不可挽回。

迭代开发

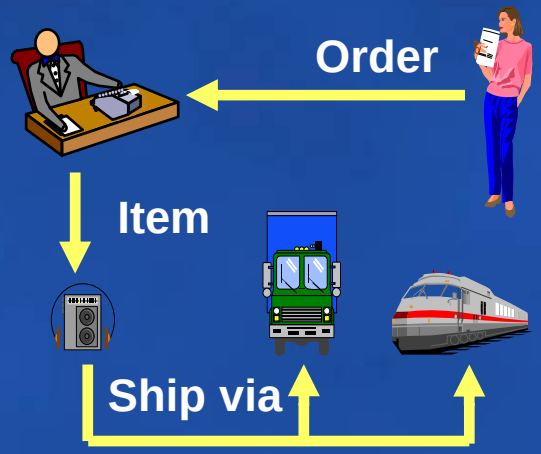


- 早期迭代解决最大的风险问题
- 每个迭代产生一个可执行的发布，其中也包含系统的所增加的功能
- 每个迭代都要进行集成、测试

迭代开发

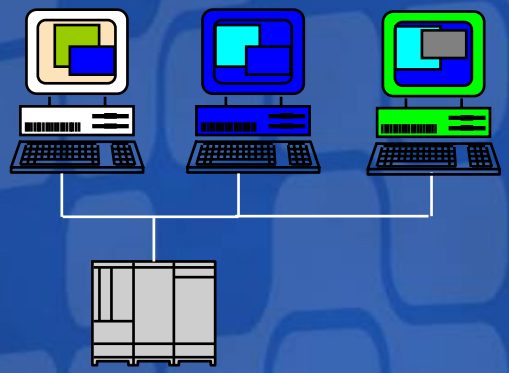
- 专注于短期目标的里程碑
- 让部分实现可以部署
- 在进一步投资之前解决主要风险
- 允许早期用户反馈
- 持续进行测试和集成

可视化建模



业务过程

“模型获取系统的关键需求.”
Dr. James Rumbaugh

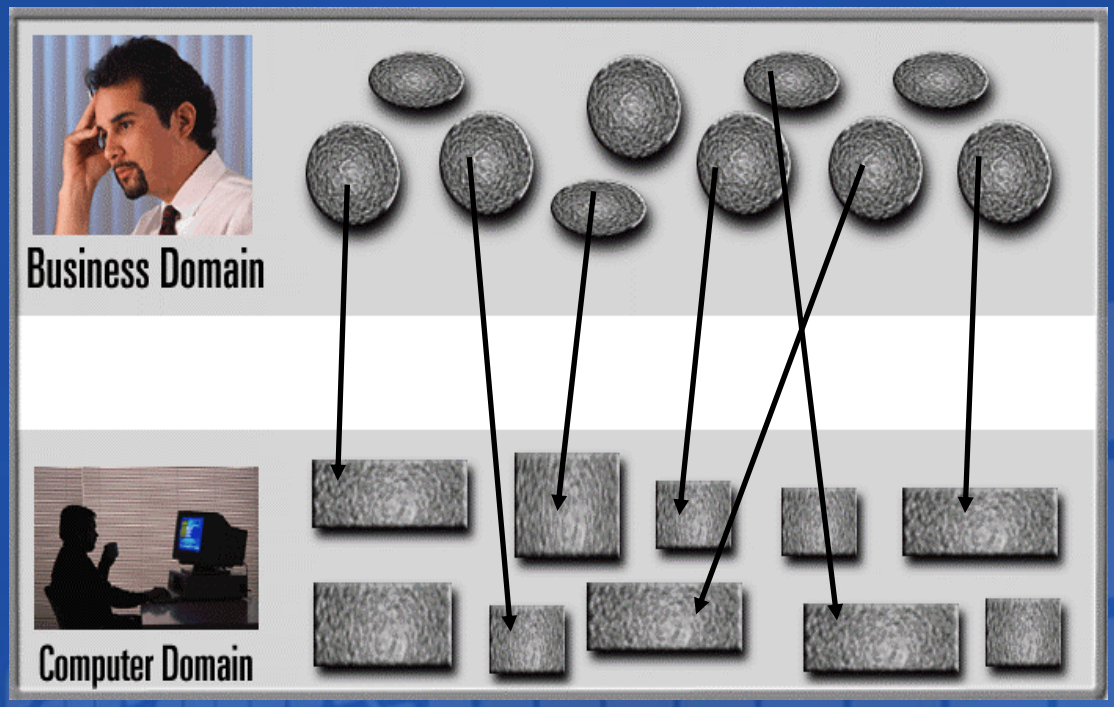


计算机系统

可视化建模使用标准的图例

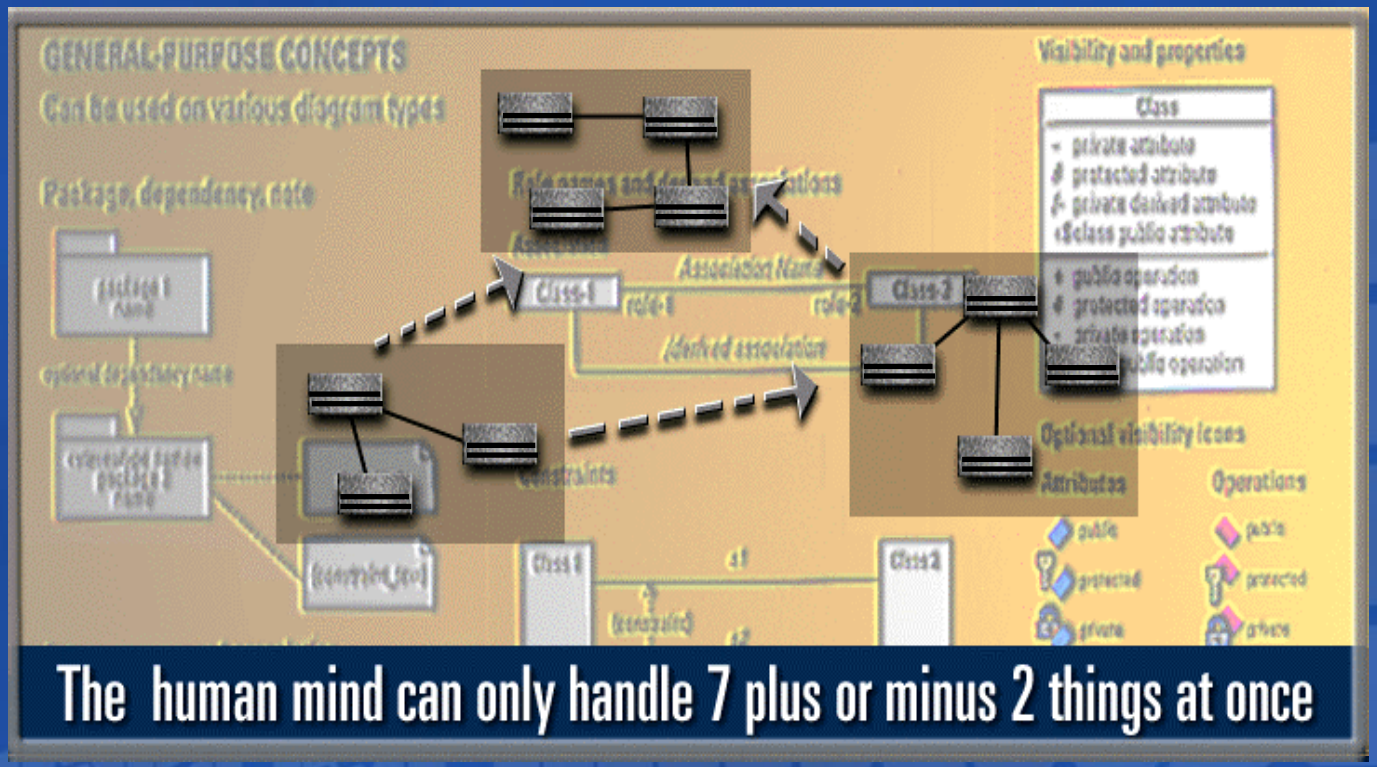
可视化模型是一种沟通工具

获取业务对象和业务逻辑



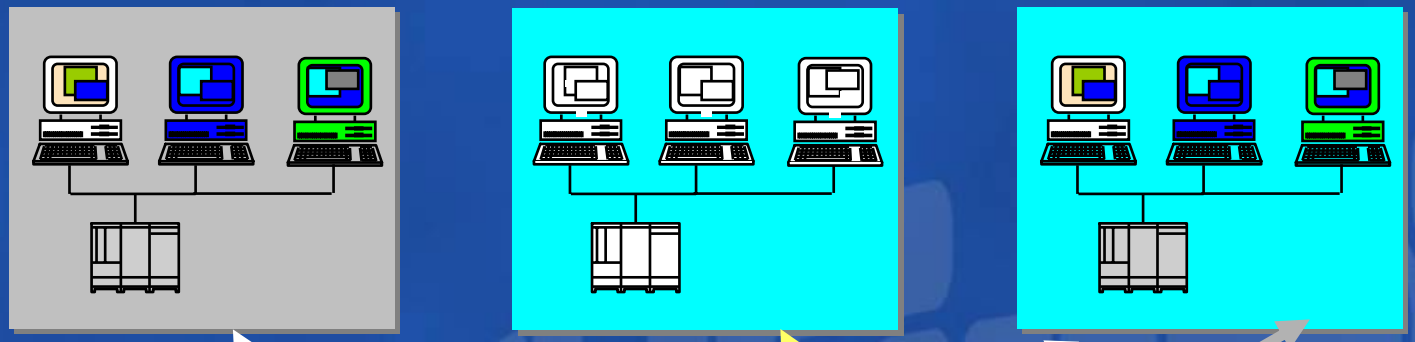
通过可视化模型分析和设计应用程序

可视化模型管理复杂性

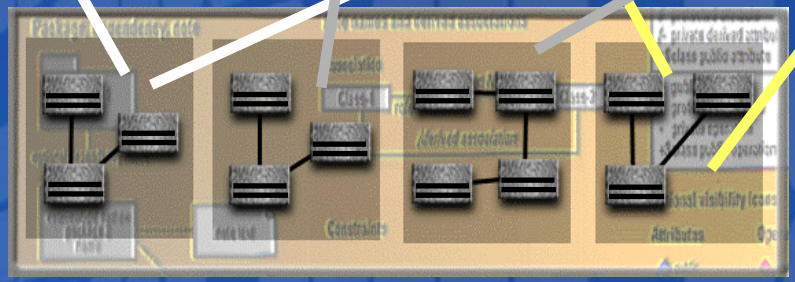

GENERAL-PURPOSE CONCEPTS
Can be used on various diagram types
Package, dependency, note
Visibility and properties
Class
Association Name
Class 1
Class 2
role-1
role-2
/derived association
Constraints
Optional visibility icons
Attributes
Operations
public
protected
private
The human mind can only handle 7 plus or minus 2 things at once

可视化模型促进重用

多个不同的应用系统



可重用组件



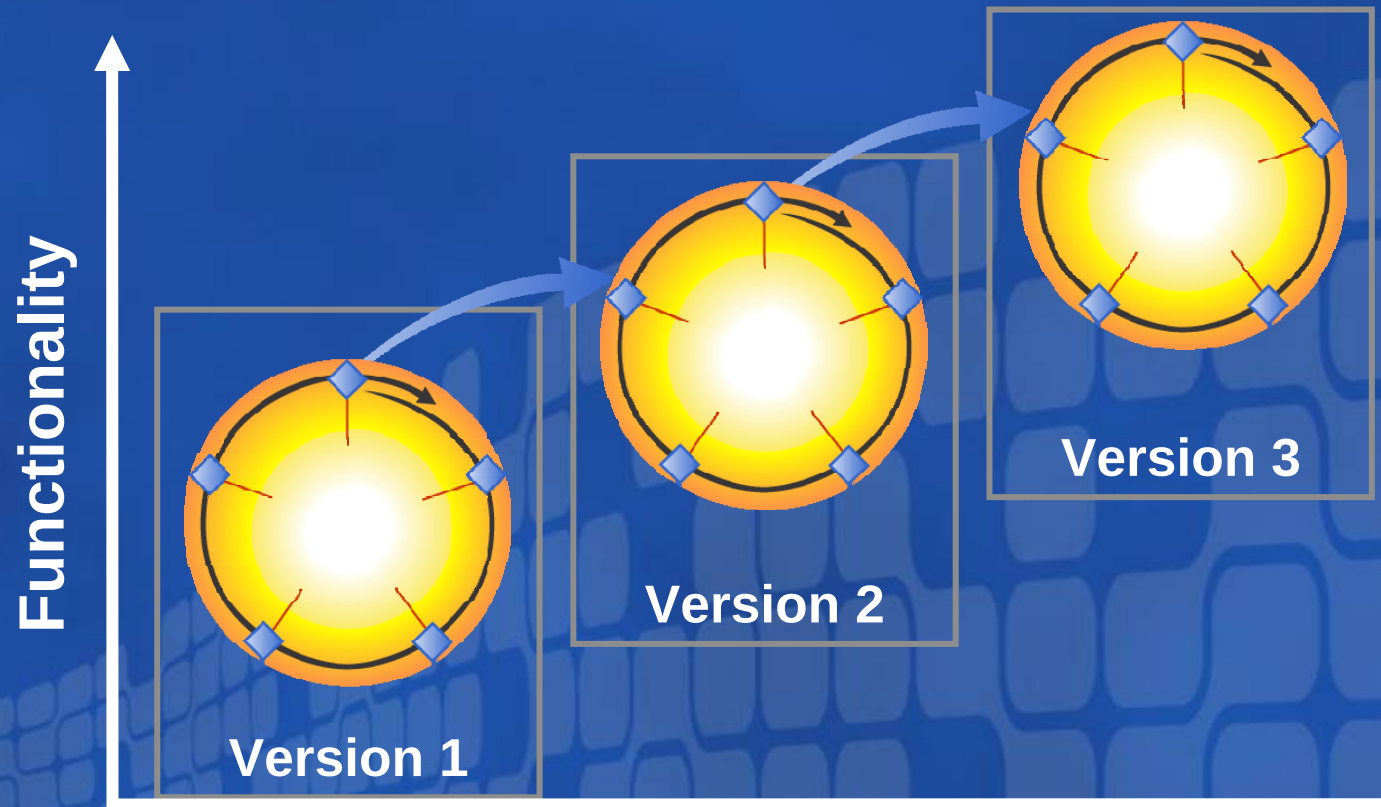
软件生命周期模型

- 瀑布模型(Waterfall)
- V模型
- 编码修正模型(Code & Fix)
- 增量模型(Incremental)
- 渐进模型(Evolution)
- 螺旋模型(Spiral)

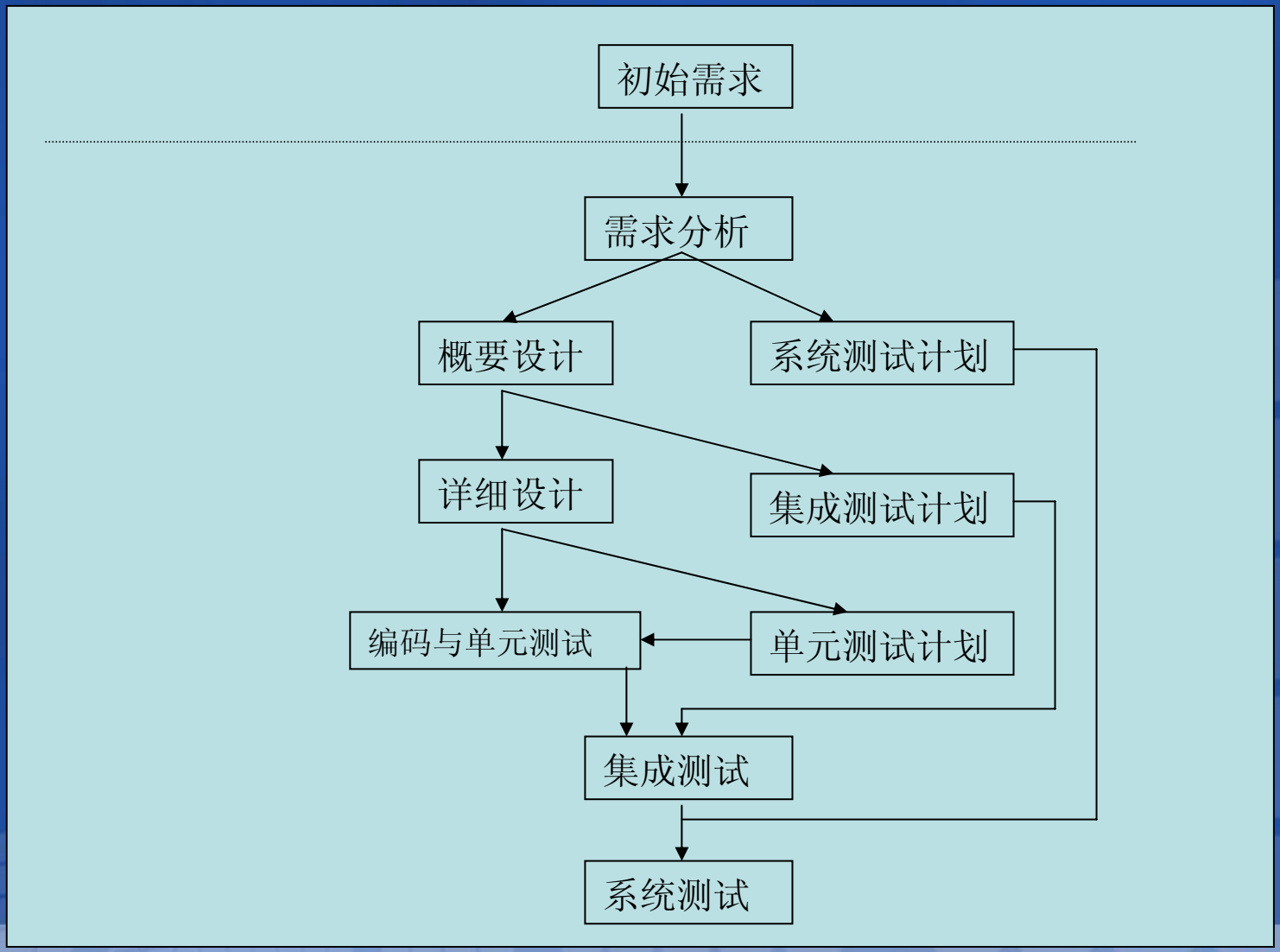
MSF 过程模型

MSF 过程模型是一种迭代方法

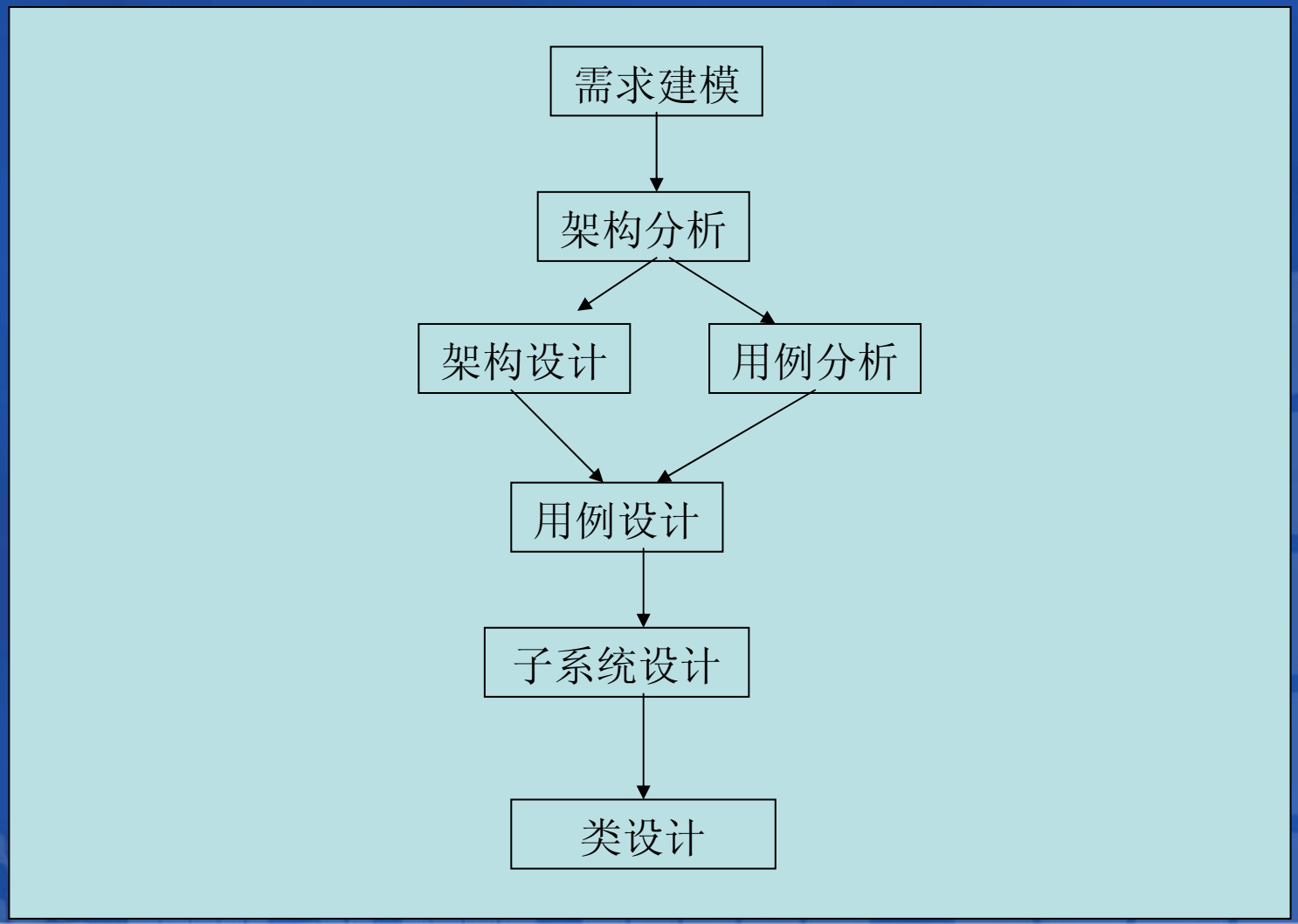
把大项目拆分成多个版本可以减小项目风险。



最常见的模型



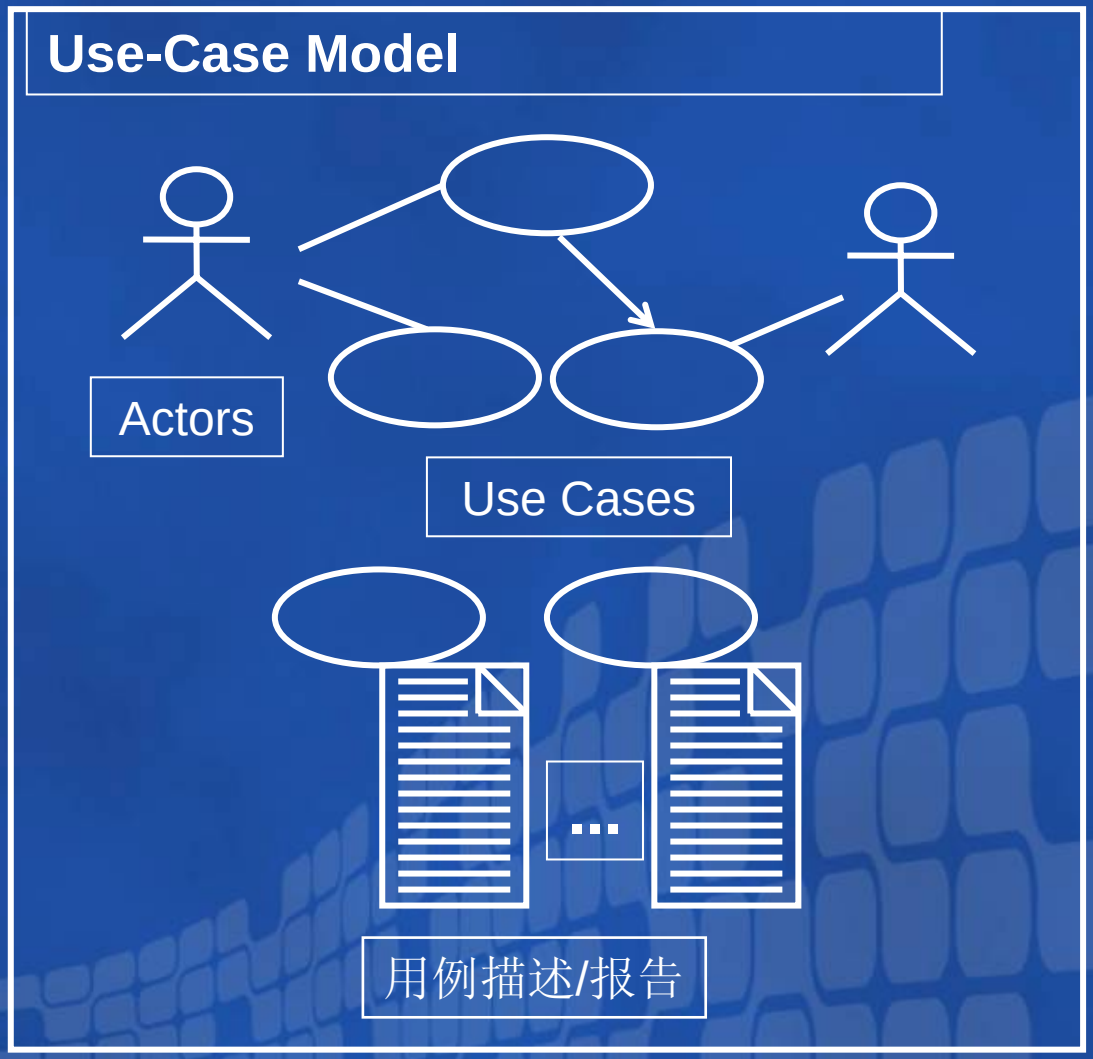
细化分析设计阶段



需求建模

- 目的
  - 系统应该做“什么”
  - 确定系统的范围
  - 为规划技术性内容的迭代提供基础
  - 定义系统的用户接口

需求制品



术语表

非功能需求规范

什么是架构分析?

- 应用程序架构师负责
- 开始尝试定义系统的各部分/块
- 为分析提出“首要原则”
- 集中在逻辑视图

架构分析的目的

- 定义系统的架构模式，关键机制和模型约定
- 定义重用策略
- 为计划过程提供输入

架构分析的过程

- 建模约定
- 分析机制
- 关键抽象
- 开始架构分层

分析机制

- Persistency
- Communication
- Distribution
- Transaction Management
- Redundancy
- Legacy Interface
- Security
- Message Routing

模式和框架

- 模式
  - 分析/设计模式
- 框架
  - 定义解决问题的一般性方法
  - 骨架结构的解决方案，其细节可能是分析/设计模式

模式的起源

- 来源于建筑学和人类学的一个概念
  - 在文化人类学中，超越个人信仰和文化差异对美感的评价是否具有一致性？
  - 是否存在一个描述我们共同认知的基础？
  - 质量是客观的吗？
  - 是什么因素让我们认为一个建筑（软件）设计是好的设计？

什么是模式？

- 建筑师Christopher Alexander上世纪70年代《The Timeless Way of Building》和《A Pattern Language》。

Patterns are solutions to a problem in a context.

什么是模式？

Each pattern describes a problem which occurs over and over again in our environment and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.

—— Christopher Alexander

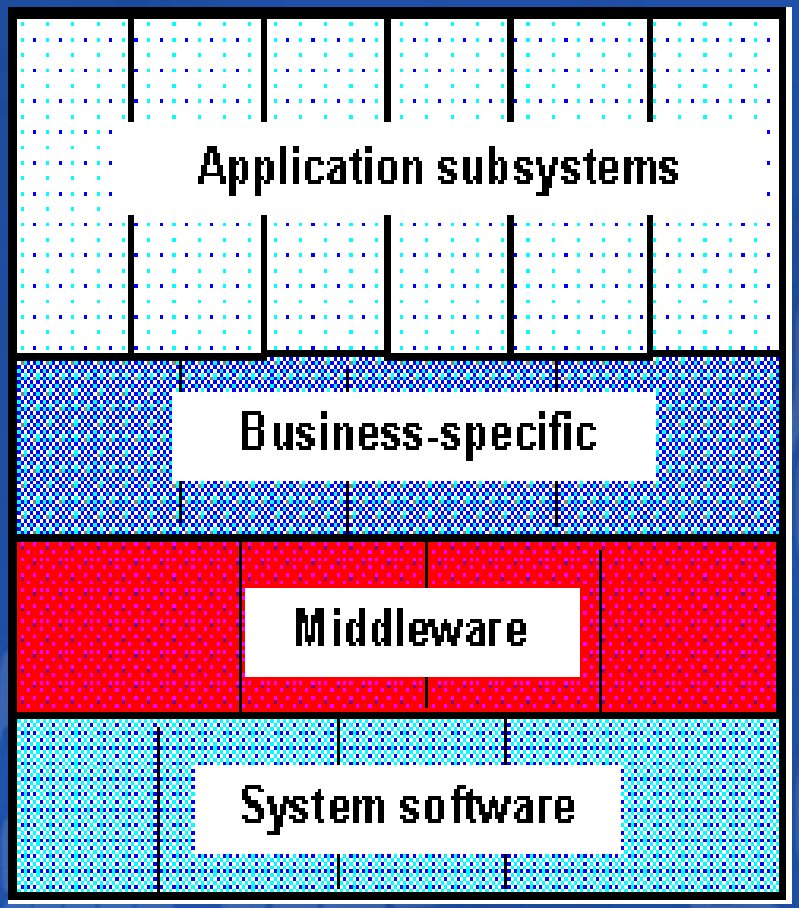
每个模式描述了一个在我们周围不断重复发生的问题，以及该问题的解决方案的核心。这样，你就能一次又一次地使用该方案而不必做重复劳动。

—— Christopher Alexander

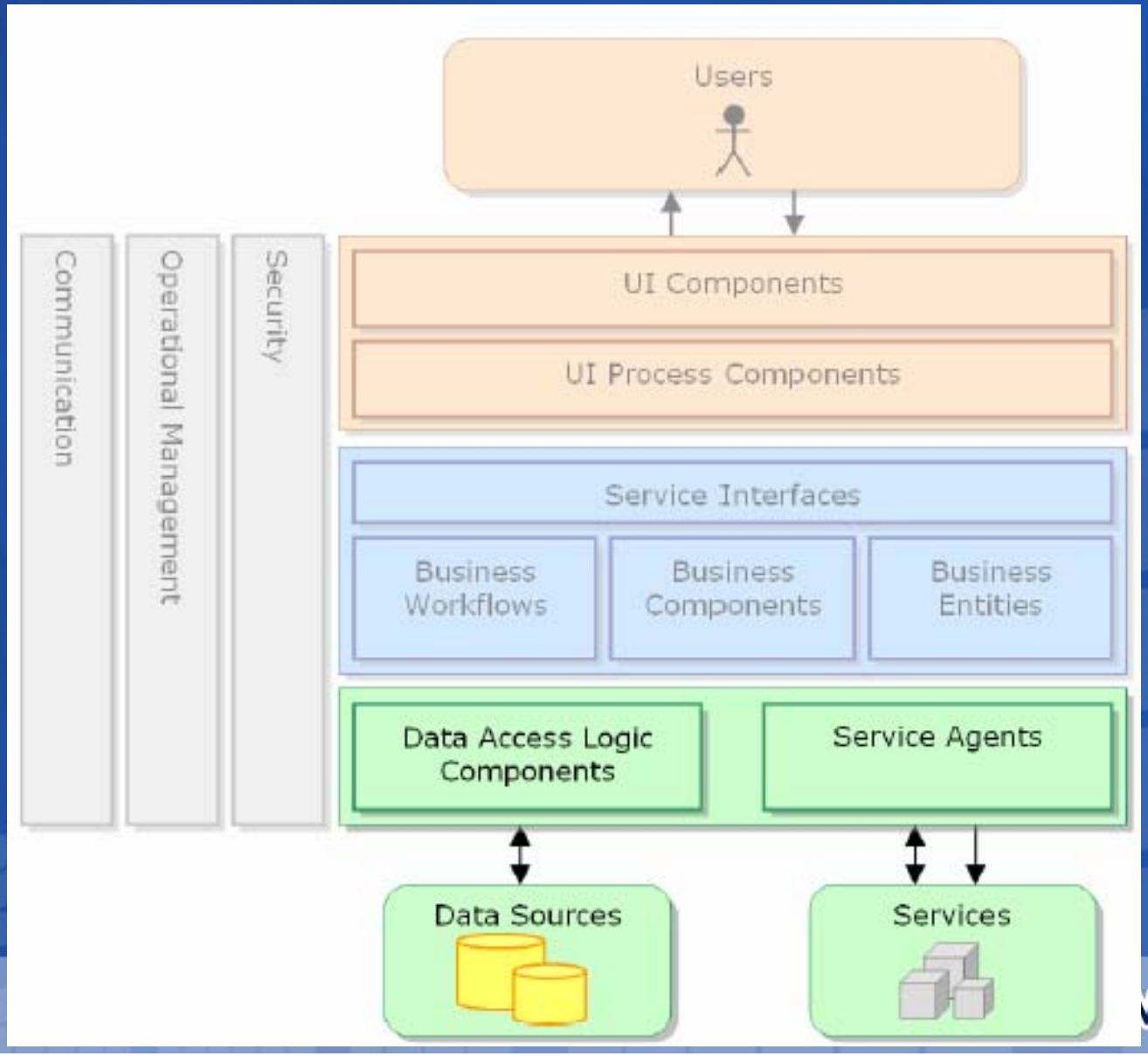
为什么需要设计模式

- 重用解决方案
  - 减少重复设计和编码
  - 规范设计和编码，便于修改和升级
- 建立通用的术语学
  - 在团队内建立对问题的通用词汇和观点
  - 便于设计阶段参考引用
  - 帮助程序员学习和团队开发
- 对软件设计的哲学思考

典型分层架构



MS Application Reference Architecture



用例分析

- 细化用例事件流
- 实现用例
  - 交互图
    - 发现分析类
    - 将用例行为分布到不同的交互模型
  - 类图
    - 添加属性和操作
    - 添加关联
  - 实现分析机制
- 统一分析类

架构设计

- 决定应用架构机制的策略
- 把分析类转换成设计类
  - 识别设计类
  - 识别子系统 + 接口
- 细化架构（通过细化分层来实现）
  - 把设计元素分配到不同的包和层中

应用架构机制

分析类	分析机制
顾客	持续，安全
计划	持续，安全
电影放映	持续，传统接口
电影	持续，传统接口
订购控制器	分布

架构机制

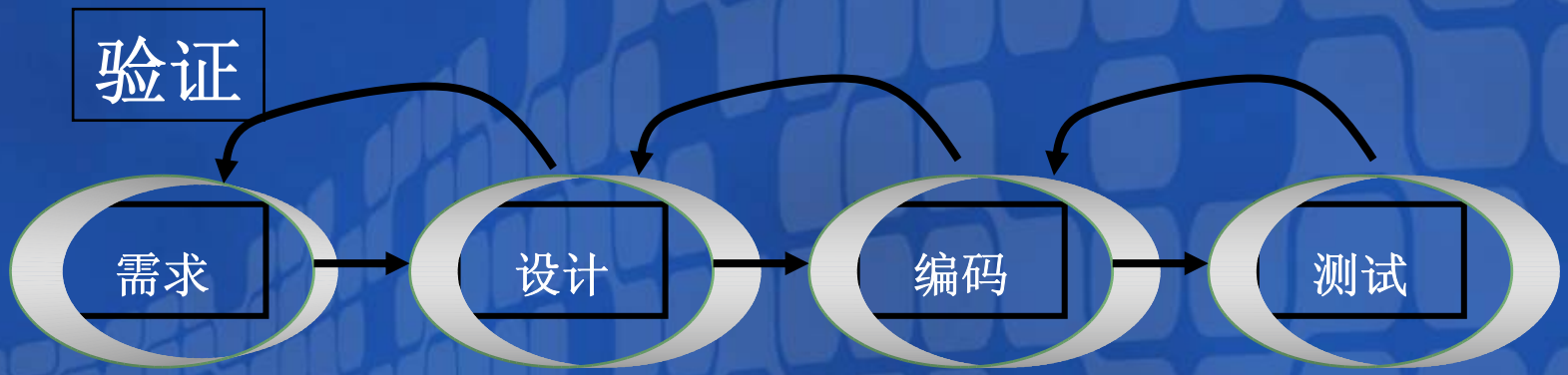
- 分析机制（Analysis Mechanism）
  - Persistency
- 设计机制（Design Mechanism）
  - RDBMS
- 实施机制（Implementation Mechanism）
  - ADO.NET

用例设计

- 将接口整合到实现中
- 识别子系统（可选－重用）
- 将设计机制整合到实现中

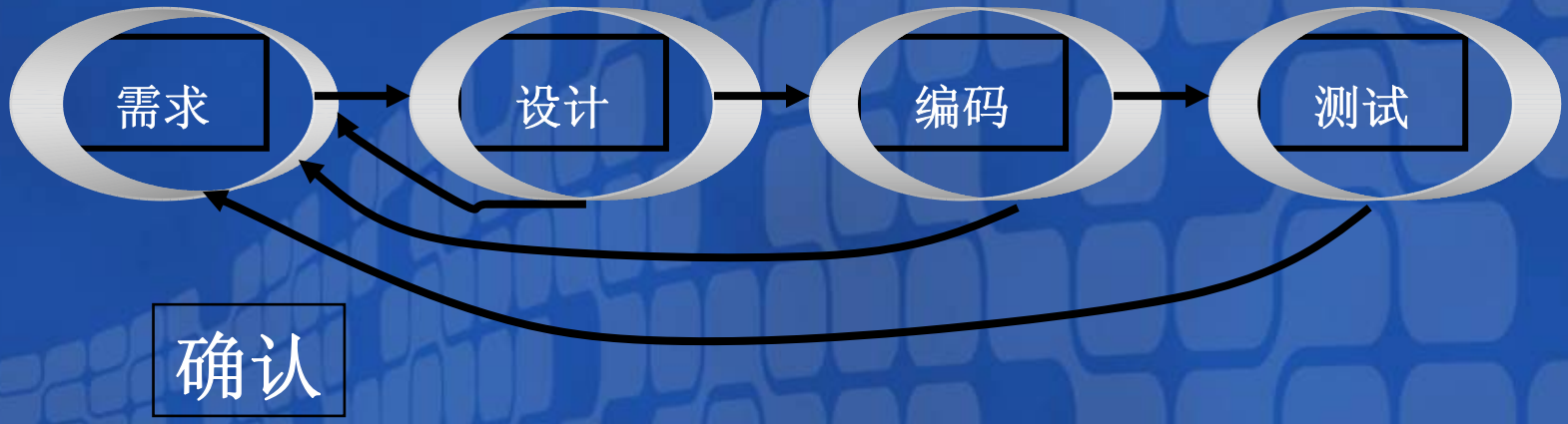
验证（Verification）

在项目的各个阶段，验证是指检测各个阶段结束时的需求规格说明RSi是否满足对该阶段的需求RDi中所提出的各项条款的过程。



确认（Validation）

在软件生存周期各个阶段，确认是指检测各个阶段结束时的需求规格说明RSi是否满足在软件生存周期初期在需求文档RD1中对该软件系统提出的各项条款的过程。

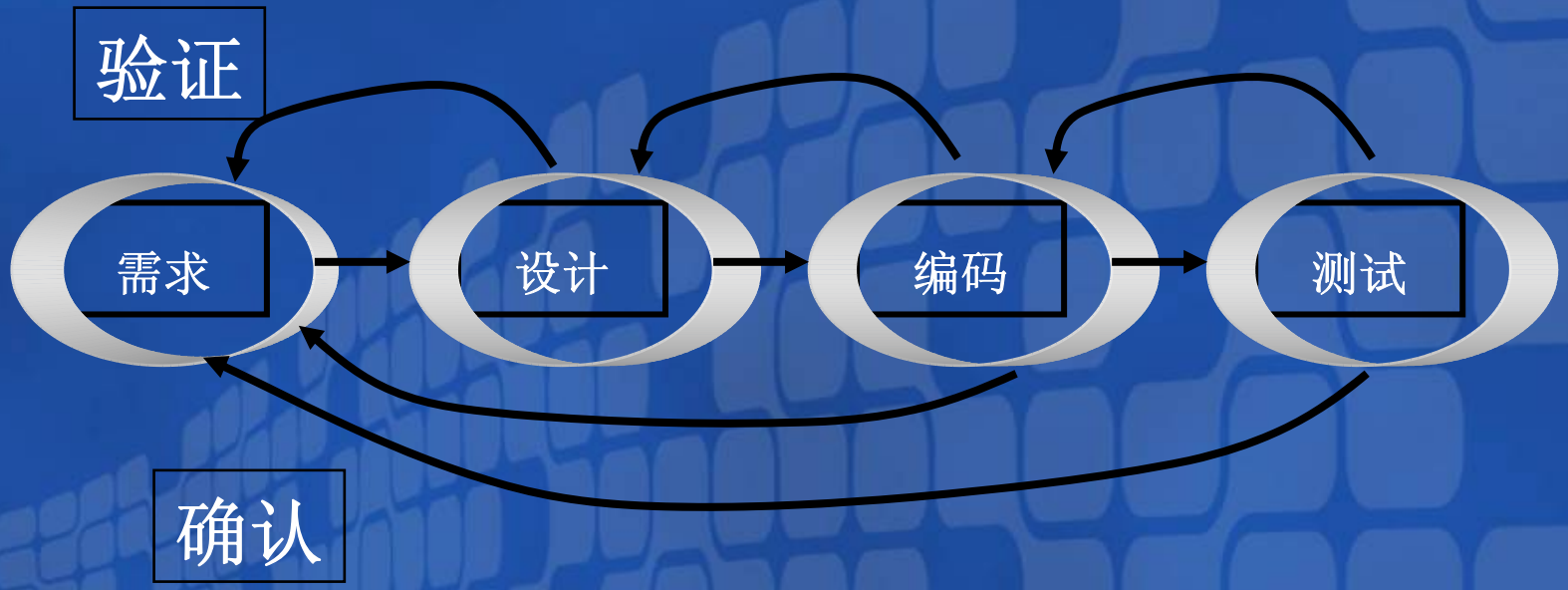


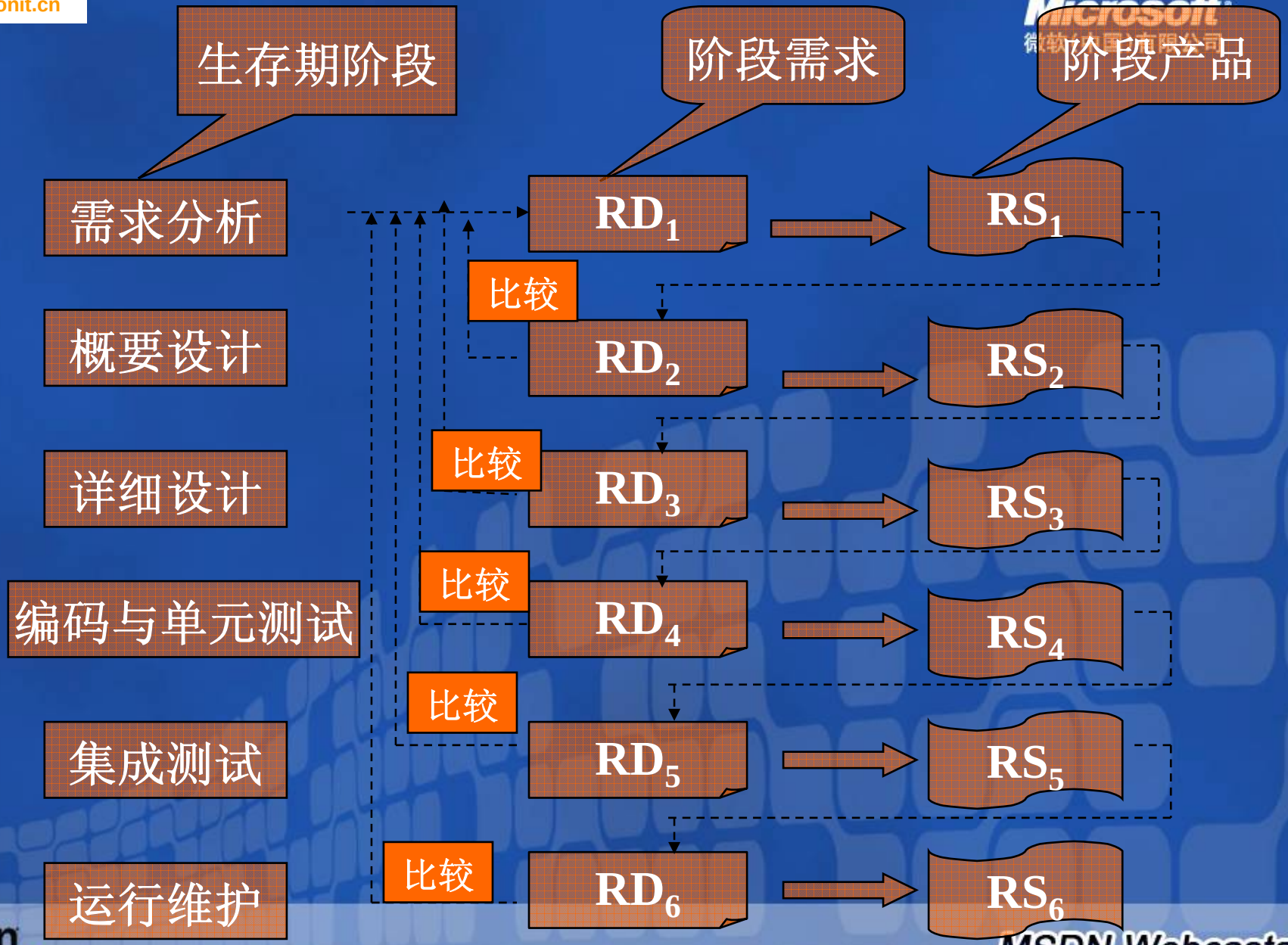
确认和验证的比较

验证是检测RSi是否与RDi相一致，

确认是检测RSi是否与RD1相一致。

软件测试既可用于验证，又可用于确认。





北京通嘉科技有限公司欢迎您！ - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Search Favorites Media

Address http://www.onit.cn/ Go Links

ONIT 通嘉科技

首页 服务 培训 产品 论坛 关于

软件开发咨询

我们为软件企业提供如下

- **开发咨询服务**
- **项目团队培养**
- **系统架构设计**

企业技术培训

- **软件架构师**
- 项目管理
- 配置管理
- 需求工程
- 分析设计
- UML
- 设计模式
- 软件测试
- .NET开发
- J2EE开发
- 数据库设计
- 数据仓库

欢迎访问
www.onit.cn

以提供现场解决问题的技术咨询服务，也可以通过电话、Email和即时通讯等方式来远程解决问题。通过这种方式能在最短时间解决项目实施过程中的关键问题，扫除技术障碍，为您的项目按期高质量完成提供坚实雄厚的保障。

项目团队培养：古人云"磨刀不误砍柴功"。对项目开发团队的前期培养是保证项目进度的一个重要手段，我们采用大型项目实战场景，为您的团队成员快速进入项目角色提供富有成效的指引，为他们提供丰富的项目经验，让项目团队成员在将来实际的项目开发过程中有效规避技术和管理陷阱，充分发挥出每个成员的各方面潜力，用其所长而避其所短，最终节省项目成本、缩短产品上市时间、提高客户满意度等等。

优秀的软件**系统架构设计**将为

我们所提供的**企业技术培训**是基于实践的培训，能让您在经过培训后立即将所学的知识用于实际工作。我们的讲师不仅具有丰富的实践经验，而且也具备丰富的教学经验，是中国教育培训界技术水平和教学水平一流的讲师。目前教育培训界很多优秀的讲师都曾从师于我们的讲师或从师于我们讲师的学生。

软件架构师培训是基于目前国内紧缺的高端软件人才而开发的一整套课程体系，其目的就是为了让您在经过培训之后，能快速地加入到中国软件架构师队伍中来，为加快国内软件业的发展助上一臂之力。该课程将以目前国内大型的软件项目为案例，让您深入了解架构设计的各个环节，是您进一步深造的必修课。

设计模式（Design Pattern）是软件设计过程中非常重要地方法和策略，可以更为有效地重用设计模型和代码，提高设计和开发效率。我们提供专门针对设计模式的

Internet

获取更多MSDN资源

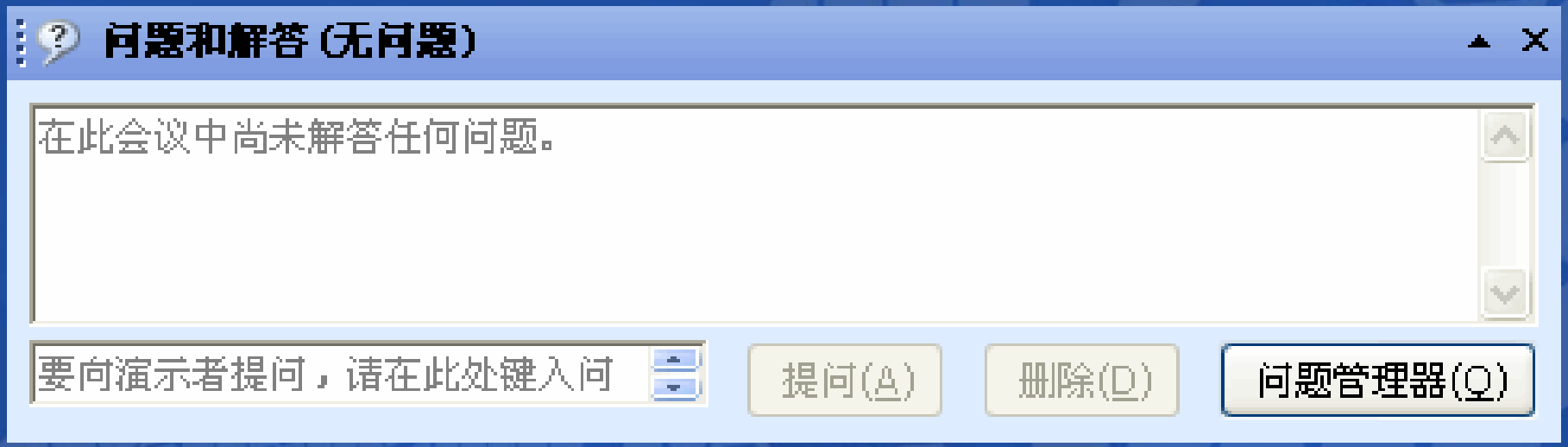
您的潜力，我们的动力

Microsoft
微软（中国）有限公司

- **MSDN中文网站**
  http://www.microsoft.com/china/msdn
- **MSDN中文网络广播**
  http://www.msdnwebcast.com.cn
- **MSDN Flash**
  http://www.microsoft.com/china/newsletter/case/msdn.aspx
- **MSDN开发中心**
  http://www.microsoft.com/china/msdn/DeveloperCenter/default.mspx

Question & Answer

如需提出问题，请单击“提问”按钮并在随后显示的浮动面板中输入问题内容。一旦完成问题输入后，请单击“提问”按钮。




您的潜力，我们的动力
Microsoft
微软（中国）有限公司
Microsoft
msdn
MSDN Webcasts