RUP/UML实践之路
——一个项目的全程回顾

胡协刚

首席软件架构师　UML/RUP专家

szjinco@public.szptt.net.cn

内容提要

- 软件开发没有银弹
- 统一通道平台开发项目简介
- 项目的开发目录结构与RUP核心工件
- 贯穿全局的统一UML模型
- 用前景文档定义目标系统
- 软件构架文档与4+1视图
- 契约式开发与单元测试
- 自动化构建与持续集成
- 迭代开发模式

软件开发没有银弹

软件项目中的常见问题

项目成员自建文件目录，文档、源码文件到处乱放，想要就随手复制一份，最后连其原始作者也分不清哪个才是最终的版本；

项目当前构架混乱、程序员各行其是、重复代码满天飞、命名到处冲突，最后只能用“一盘散沙”来形容开发出的软件；

开发人员老是理解错对方的含义，返工成了家常便饭；

软件项目中的常见问题

开始时好像一切都很顺利，几十个开发人员很快就完成了各自的编码，然而到了集成时，却总是有没完没了的bug，更要命的是集成编译也常常通不过；

张三这几天老是抱怨他调试了三天才解决的一个bug，竟然是因为用的一个库被别人修改了，偏偏就忘记了通知他；

眼看客户要求的发布期就要到了，项目经理汇报说："90%的功能已经完成了，就差ABCD几个功能还没动，不过其中有两个是关键功能"，于是向客户交付一个能满足他们基本需求的版本的可能性也没有了；

软件项目复杂、不确定和高风险

- 软件因其固有的复杂、不确定和高风险等属性，使得其开发活动非常难以控制
- 软件产品用于解决一个或多个领域的现实问题，不仅与其开发者本身直接相关，还牵涉到客户、最终用户、第三方伙伴等众多涉众，所谓众口难调，但一个成功的软件必须能满足多方面的需求
- 软件所依赖的核心技术、项目的开发方法、软件过程、项目管理和团队协作等，都是事关项目成败的关键因素

CMMI与RUP

- 根据CMMI的定义，软件过程要达到第三成熟度等级，需要实施十八个关键过程域；普通团队虽然不一定向此标准看齐，但实际上仍然涉及到这十八个关键过程域所涵盖的近百项活动，只不过可能实施的力度较小、质量不高、或者忽略了活动本身的含义而已
- RUP与CMMI第三成熟度等级相对应，定义了九个核心门类（Discipline），和几百项活动
- 这还仅仅只是软件过程的范畴，项目中人的管理、沟通以及具体的关键技术等等，其牵涉面将更广

软件开发没有银弹

- 诸多因素分属不同的范畴，针对不同的问题，需要应用不同的技术、技能和方法，要求项目成员拥有不同的知识、素质与背景
- 没有一种方法或途径能够解决软件项目中所有的问题，即所谓“没有银弹”，所以不要迷信存在所谓的一劳永逸的终极解决方案

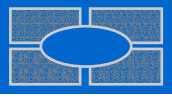
成功意味着多方面的艰苦努力

- 软件项目适用于“木桶原理”，要获得成功，必须由所有成员在多个方面都付出艰苦的努力，包括完成一些繁琐与枯燥的工作，并保证任何一个环节都不出重大问题
- 在下面介绍的一个实际项目中，将会看到我们是如何通过裁减RUP过程来主导项目的开发活动，使用统一的UML模型来表达和沟通设计，利用共享的产品目录结构来协同开发，并通过加强团队建设，扬长避短，发挥各成员的长处等——总之从多个方面来努力，方才最终达成项目成功的过程

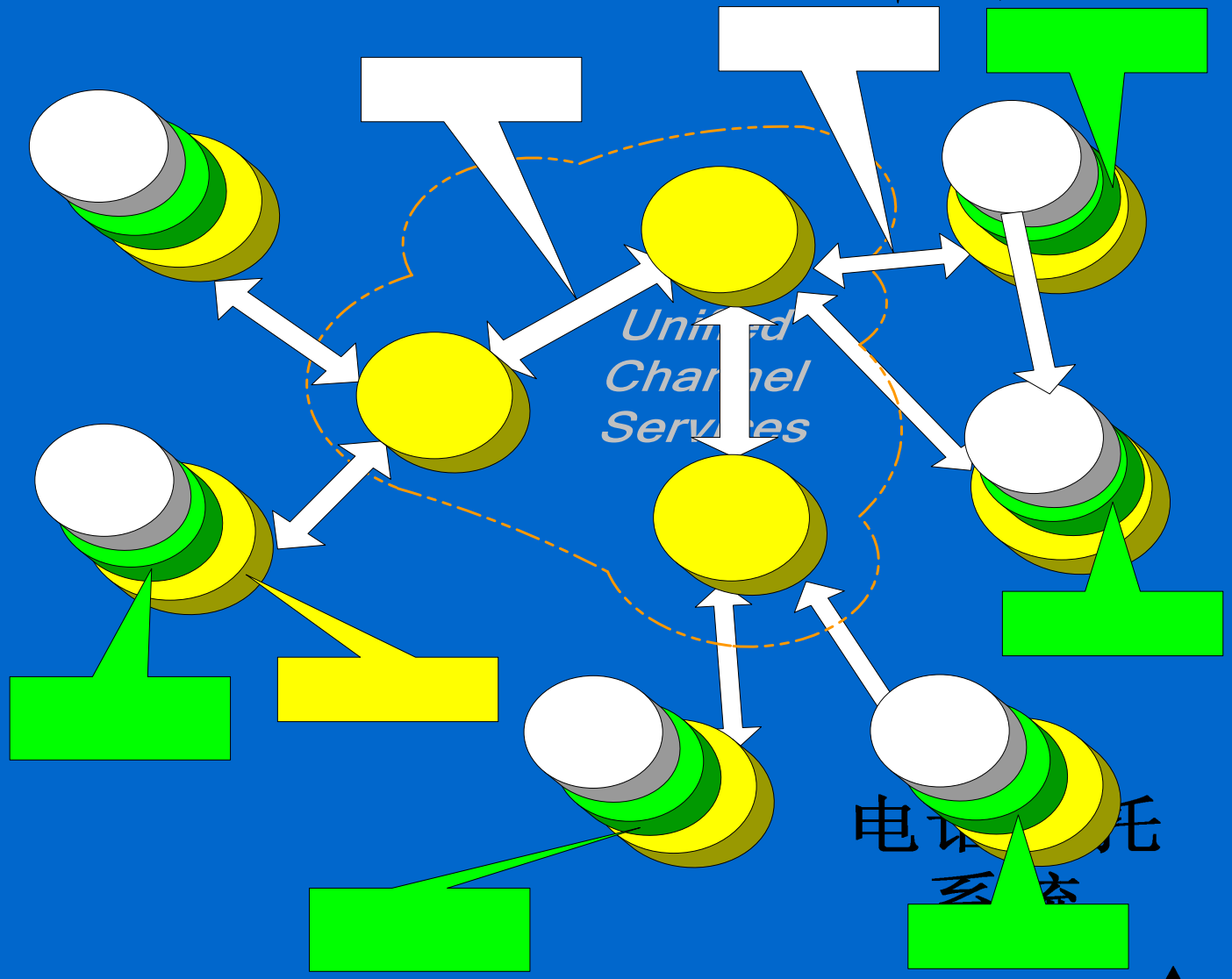
实例项目简介

证券统一通道平台项目

项目的目标系统（统一通道平台）是在证券公司总部和所有营业部网点统一部署的、适应不同网络拓扑结构的、支持内外网段物理安全隔离的、基于消息的通讯中间件平台系统，用以支持热自助、电话委托等各类外围客户端系统，集中经纪业务系统、传统营业部柜台系统等核心服务系统，以及银行端银证业务系统等第三方系统的透明接入



UCS 辐射型拓扑部署结构



项目开发过程概貌

- 项目采用RUP的迭代开发生命周期模型，总计经历了六次迭代，持续约八个月，项目组成员3~6人，耗费共计685个工作人日，最终有效代码30,471行，千行代码集成缺陷率低于1
- 使用Clearcase UCM实施配置管理，并基于此组织项目的产品目录结构，囊括了RUP定义的主要核心工件
- 使用Rose开发了统一的UML模型，从上下文分析、功能用例模型、到设计模型、进程模型、部署模型等，涵盖了项目大部分分析、设计成果，利用了Rose的正向工程生成大部分代码框架
- 目标系统的分析、设计、编码完全实现面向对象

项目开发过程概貌continue

- 项目以软件构架文档为中心，配合Rose模型、详细设计文档，使得产品的设计比较到位
- 项目引入了契约式编程方法，编码语言为标准c++，使用CppUnit测试框架进行较为广泛的单元测试，对产品的质量改进帮助很大
- 使用Ant+Cpptasks工具初步实现了自动化构建与集成
- 项目本身系公司的过程改进试点项目，集中了部门最优秀的人员，大家都表现了良好的团队协作精神，建立了密切的私人关系
- 作为试点项目，其成果，已经作为模板工程在公司大力推广

项目的开发目录结构

开发（产品）目录结构

- 开发目录结构是关系到软件项目健康运作的关键因素；
- 开发（产品）目录提供了项目团队进行开发、管理等活动的统一共享场所，它需要满足不同涉众/角色（分析、设计、测试、管理、支持等人员），在不同的阶段（启始、精化、构建、迁移、维护等），对不同类型工件（源码、文档、模型、安装包等）进行访问的多种场景需求；
- 开发目录的结构本身是静态的，但在整个项目生命周期中，其所存放工件内容以至组织结构却在不断地变化，因此它也具有动态的特性。

目录结构的组织原则

- 目录必须采用一个明确的工件分类方案，帮助使用者将工件存放到正确的地方，同时方便使用者迅速找到或定位他所需要的工件；
- 目录应当避免出现冗余的工件副本，并且像软件一样具备良好的层次结构，它应当最充分地体现分而治之的组织原则；
- 同一目录内容高内聚、不同目录依赖低耦合，从而帮助使用者将注意力集中于当前的工作上；
- 目录与工件都必须受控，这种控制既针对其静态结构，又要涵盖其动态的演变过程，并确保其历史版本记录和基线的备份。

软件配置管理

所谓配置管理是指——在整个软件生命周期过程中对软件的配置及其变更进行管理和控制；（即对开发目录内容的动态变化进行控制）

配置管理的目标有——

- 建立和维护软件产品在整个项目生命周期中的完整性和可跟踪性
  - ✓所有配置项（工件）被标识出来
  - ✓所有的变更都是受控的
  - ✓过程可跟踪
- 支持并行开发（提供团队共享工作环境）
- 提供项目管理的基础
- 最大限度地减少错误，提高生产率

置于配置管理之下的目录结构

- 开发（产品）目录结构的规划是软件配置管理的主要内容，由过程工程师、构架师配合配置经理来共同制定，并在配置工具的支持下予以实施；
- ClearCase Stream（流）的划分——

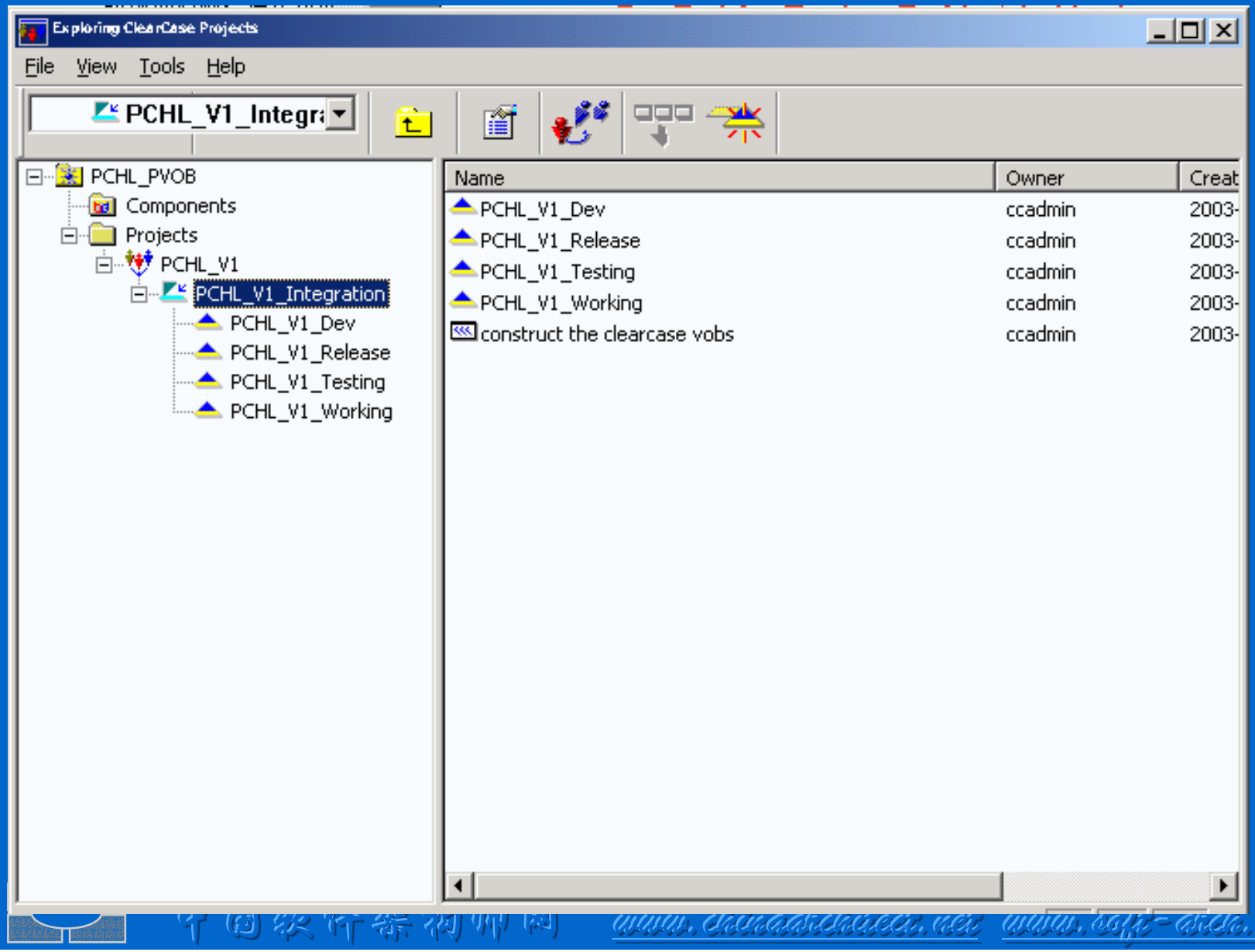
- PCHL_V1_Integration	整个项目所有工件的最终整合场所，用于项目级基线构建、进行集成和确认集成测试
--- PCHL_V1_Working	需求、设计、管理、过程等工作场所
--- PCHL_V1_Dev	项目所有构件的开发场所
--- PCHL_V1_Testing	测试组的测试场所
--- PCHL_V1_Release	发布演示场所

置于配置管理之下的目录结构

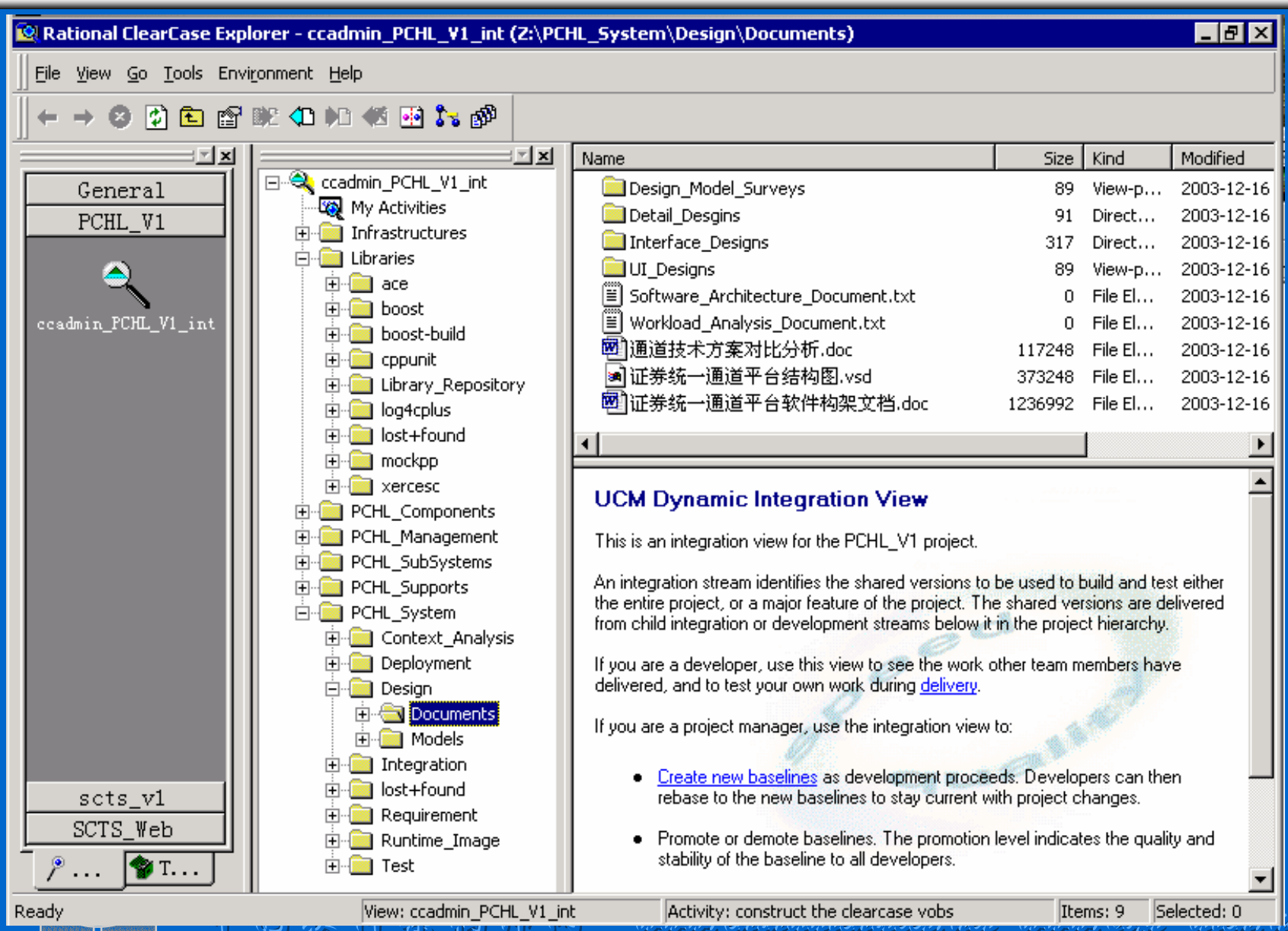
- 项目目录参照RUP的工件集来组织
- Clearcase VOB的划分——

Infrastructures	内部开发的相对独立、复用度强的基础应用包，针对第三方产品进行封装的调用接口等
Libraries	从组织外部获取的源码库、开发组件等，主要来自于开放源码；
PCHL_Components	项目组针对目标系统开发的构建于底层开发包、基础设施之上的所有构件
PCHL_SubSystems	构建于底层开发包、基础设施与所有构件之上的可执行子系统（或系统），通常是目标系统的最终交付实体
PCHL_System	项目目标系统主体工件目录，包含需求、设计、集成、测试等
PCHL_Management	项目的管理工件目录
PCHL_Supports	项目的过程、环境、标准等支持工件目录

ClearCase UCM工程示例



开发目录结构示例


Rational ClearCase Explorer - ccadmin_PCHL_V1_int (Z:\PCHL_System\Design\Documents)
File View Go Tools Environment Help
General
PCHL_V1
ccadmin_PCHL_V1_int
scts_v1
SCTS_Web
ccadmin_PCHL_V1_int
My Activities
Infrastructures
Libraries
ace
boost
boost-build
cppunit
Library_Repository
log4cplus
lost+found
mockpp
xercesc
PCHL_Components
PCHL_Management
PCHL_SubSystems
PCHL_Supports
PCHL_System
Context_Analysis
Deployment
Design
Documents
Models
Integration
lost+found
Requirement
Runtime_Image
Test
Name
Size
Kind
Modified
Design_Model_Surveys 89 View-p... 2003-12-16
Detail_Desgins 91 Direct... 2003-12-16
Interface_Designs 317 Direct... 2003-12-16
UI_Designs 89 View-p... 2003-12-16
Software_Architecture_Document.txt 0 File El... 2003-12-16
Workload_Analysis_Document.txt 0 File El... 2003-12-16
通道技术方案对比分析.doc 117248 File El... 2003-12-16
证券统一通道平台结构图.vsd 373248 File El... 2003-12-16
证券统一通道平台软件构架文档.doc 1236992 File El... 2003-12-16
UCM Dynamic Integration View
This is an integration view for the PCHL_V1 project.
An integration stream identifies the shared versions to be used to build and test either the entire project, or a major feature of the project. The shared versions are delivered from child integration or development streams below it in the project hierarchy.
If you are a developer, use this view to see the work other team members have delivered, and to test your own work during delivery.
If you are a project manager, use the integration view to:
Create new baselines as development proceeds. Developers can then rebase to the new baselines to stay current with project changes.
Promote or demote baselines. The promotion level indicates the quality and stability of the baseline to all developers.
Ready
View: ccadmin_PCHL_V1_int
Activity: construct the clearcase vobs
Items: 9
Selected: 0

项目的主要文档

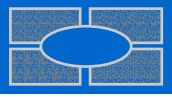
参照RUP核心工件，根据项目需要确定在项目中开发和使用如下文档——

- 《证券统一通道平台前景文档》
- 《证券统一通道平台补充规约》
- 《证券统一通道平台软件构架文档》
- 《XXXX详细设计文档》
- 《软件开发计划》
- 《测试计划》
- 《系统性能测试报告》
- 《系统安装与配置手册》
- 《外围服务协议API应用开发手册》

统一的UML模型

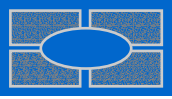
OMG的模型驱动架构

- OMG主导的MDA（Model Driven Architecture）正在成为下一代软件开发的主流模式
- 基本的模型转换关系：Computation Independent Model (CIM)-> Platform Independent Model (PIM)-> Platform Specific Model (PSM)-> Implementation

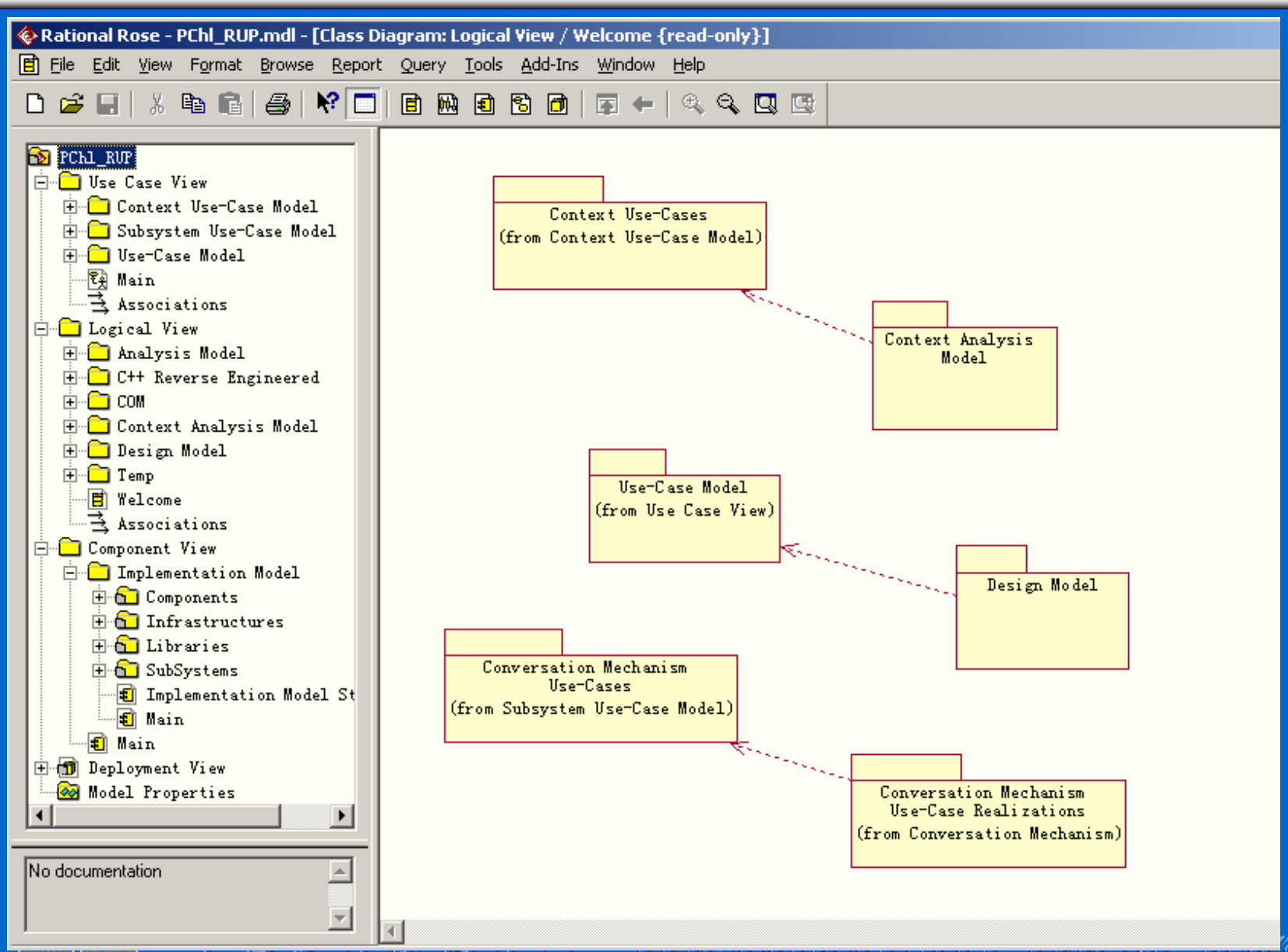


贯穿全局的统一UML模型

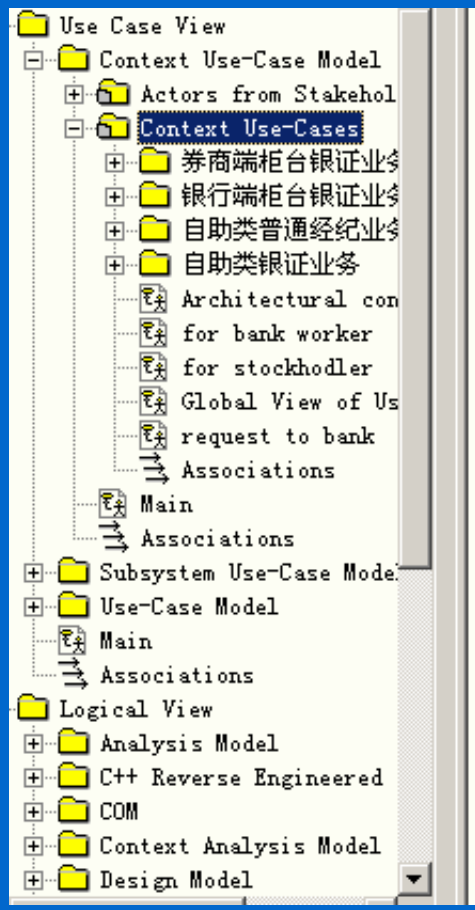
- 使用一个统一的可视化模型来表达项目的分析、设计思想，进而通过标准的语言（UML）来进行成员间的沟通，以减低传递过程中信息丢失和错误理解的风险
- 利用建模工具（Rose）对双向工程（Round Trip Engineering）的支持，初步实现MDA中平台相关的模型到实施代码框架的转换（ PSM-> Implementation）

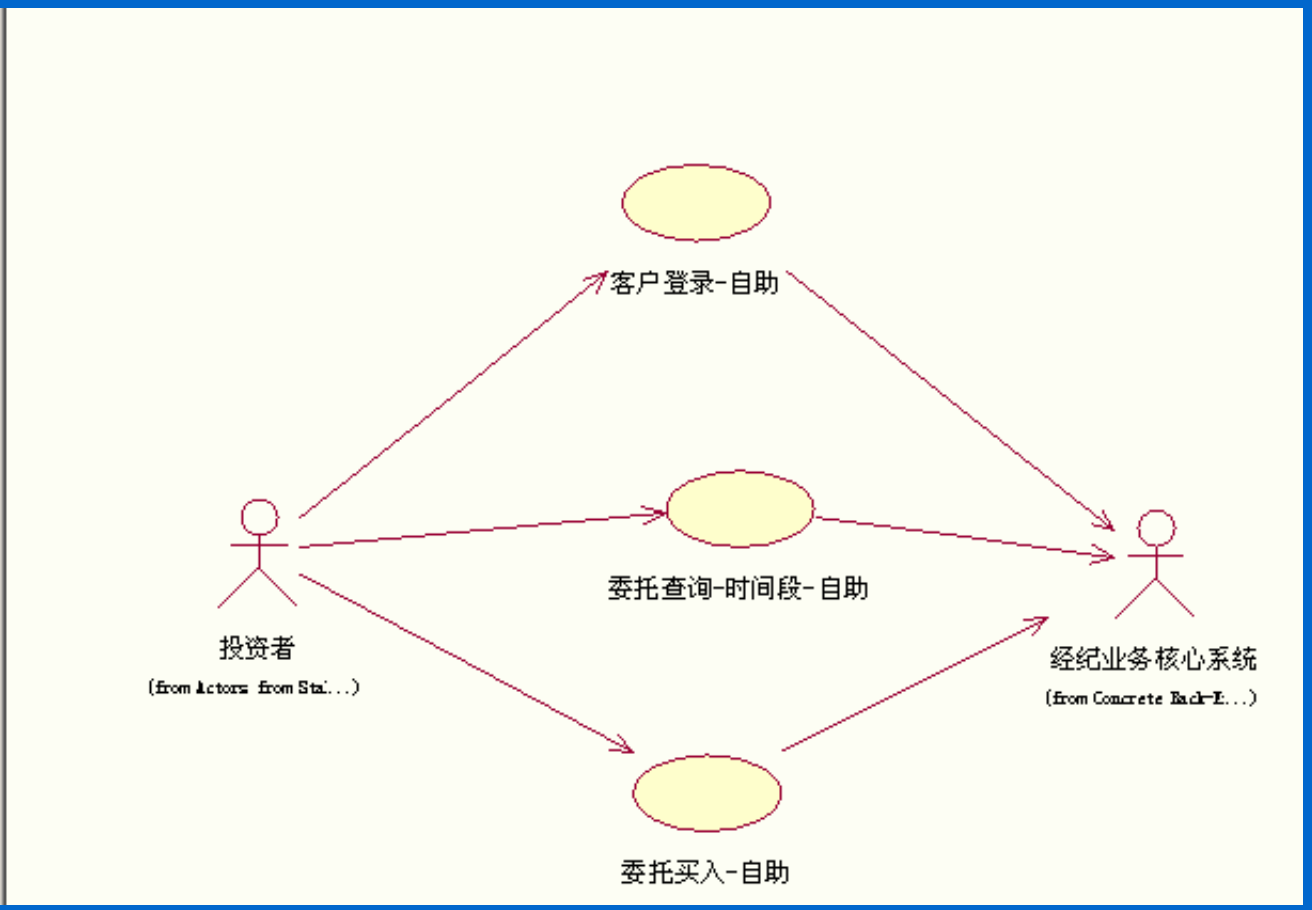


统一的Rose模型示例

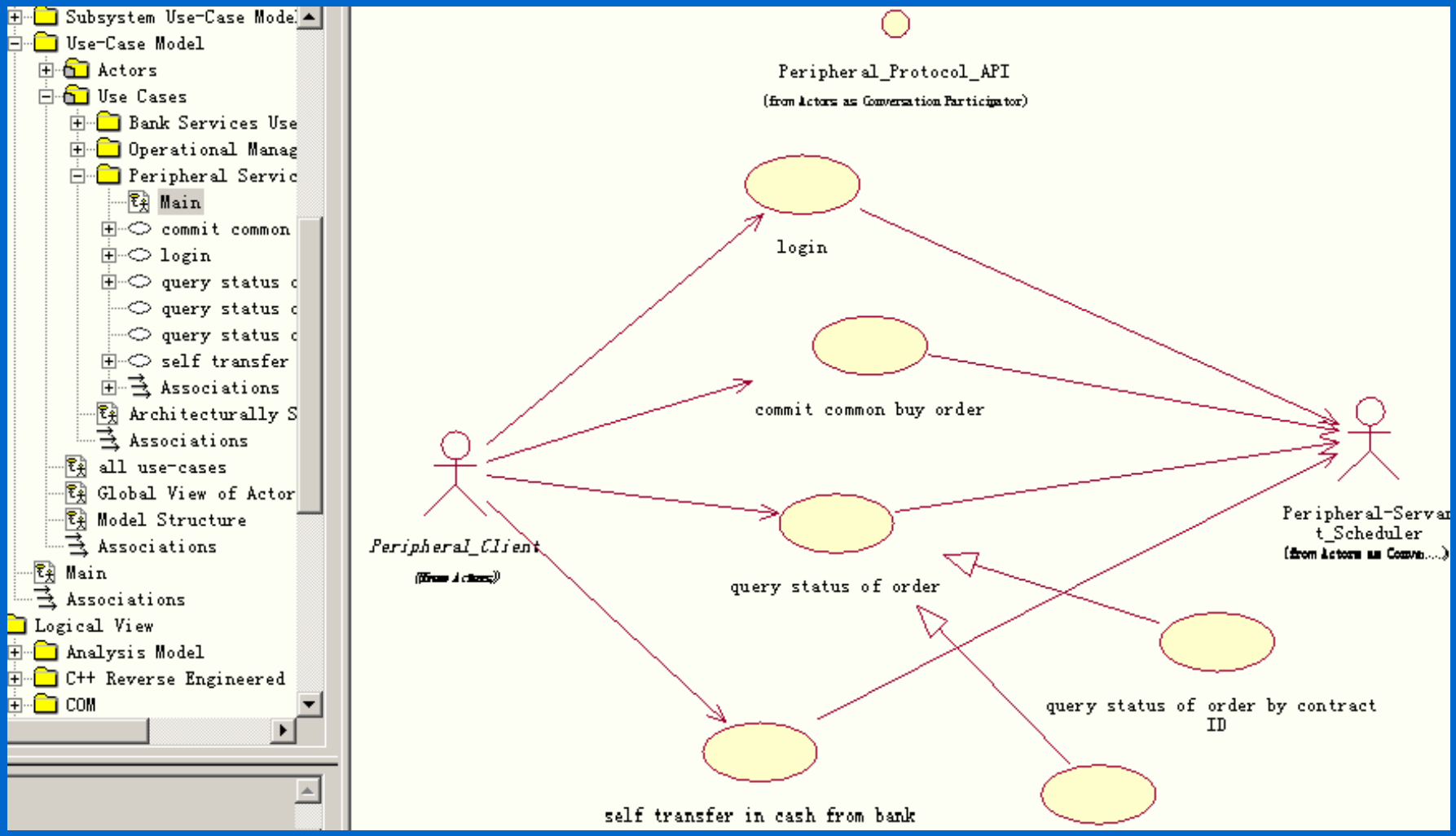


上下文分析模型示例

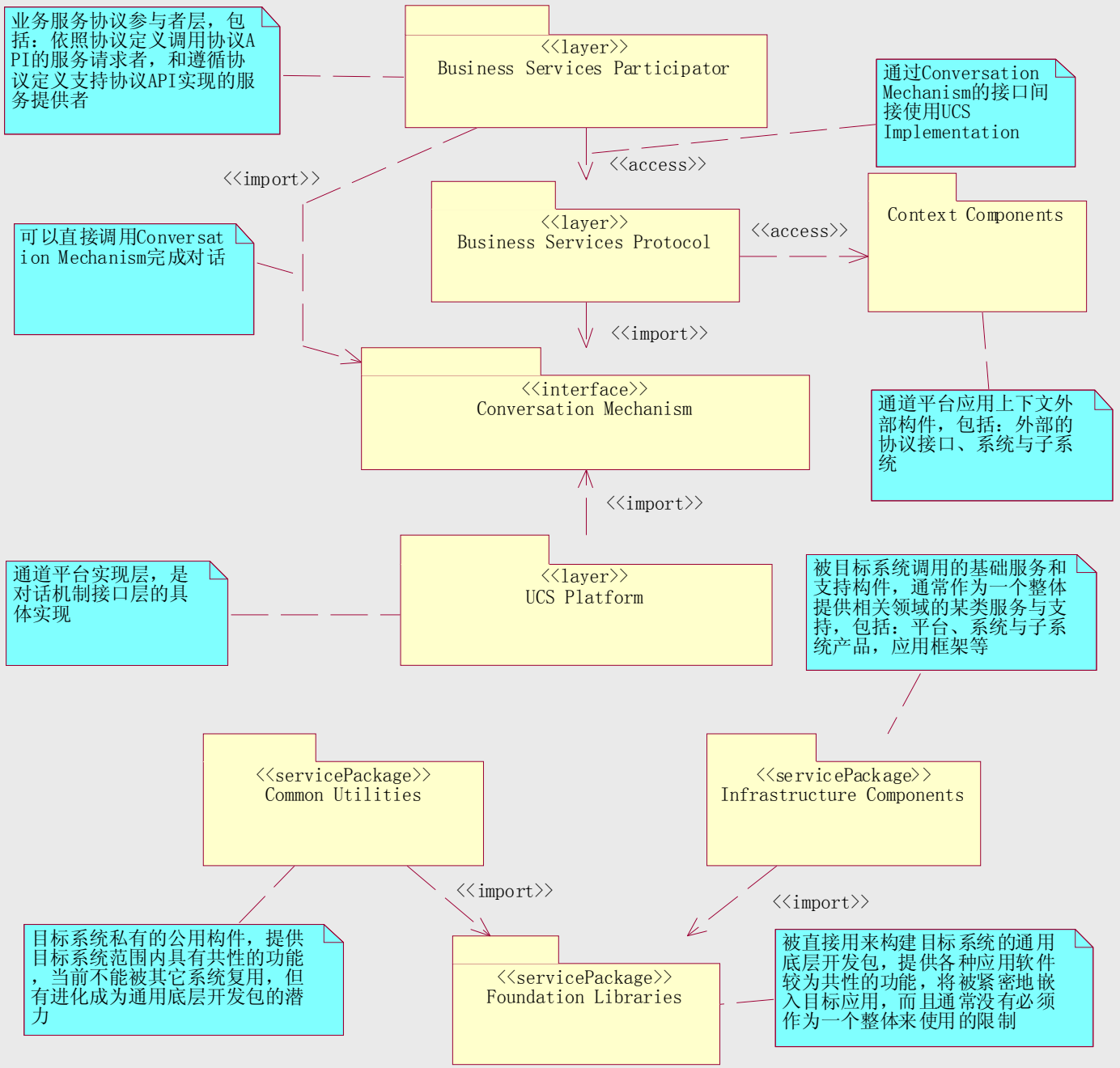


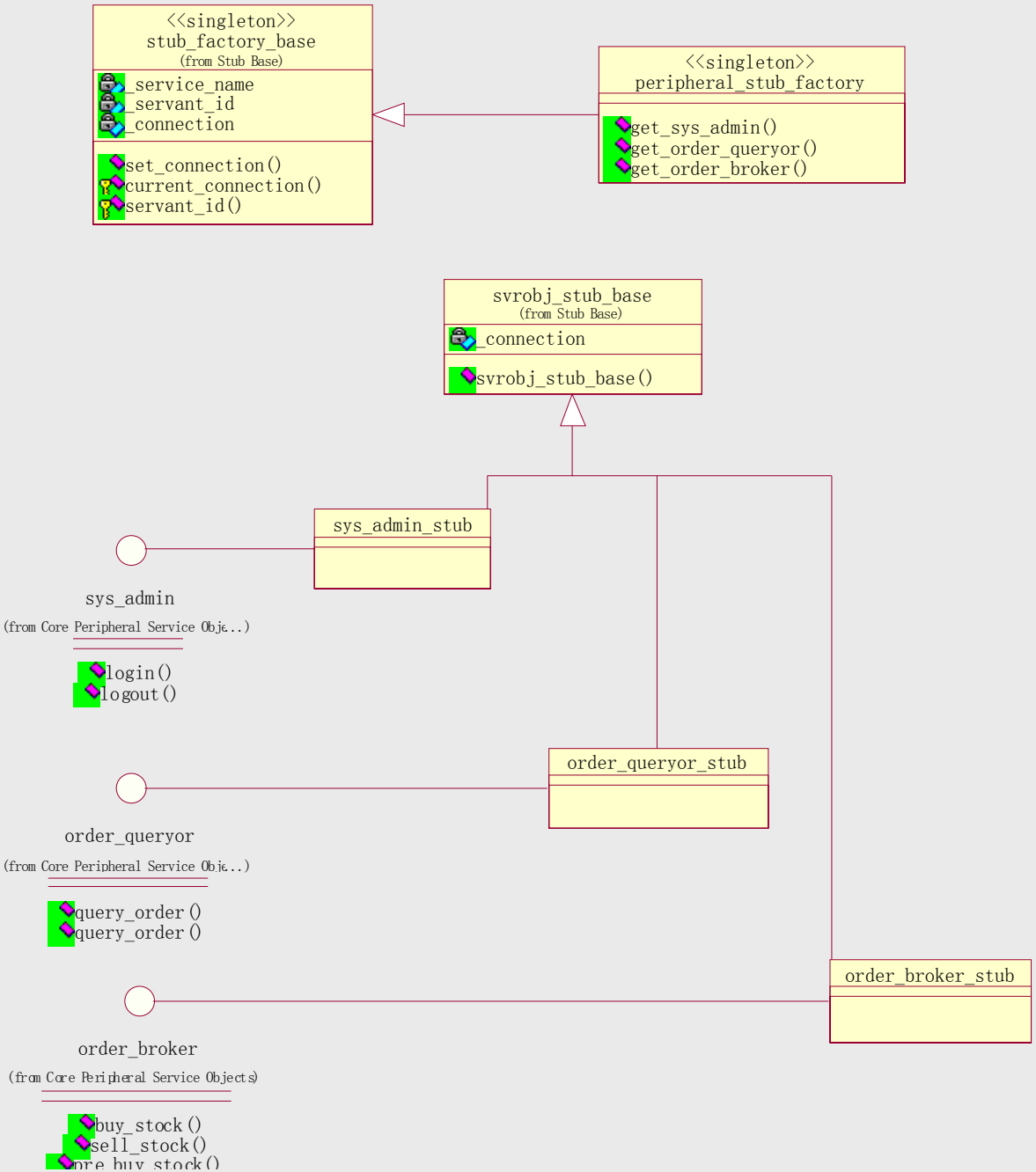


需求模型示例



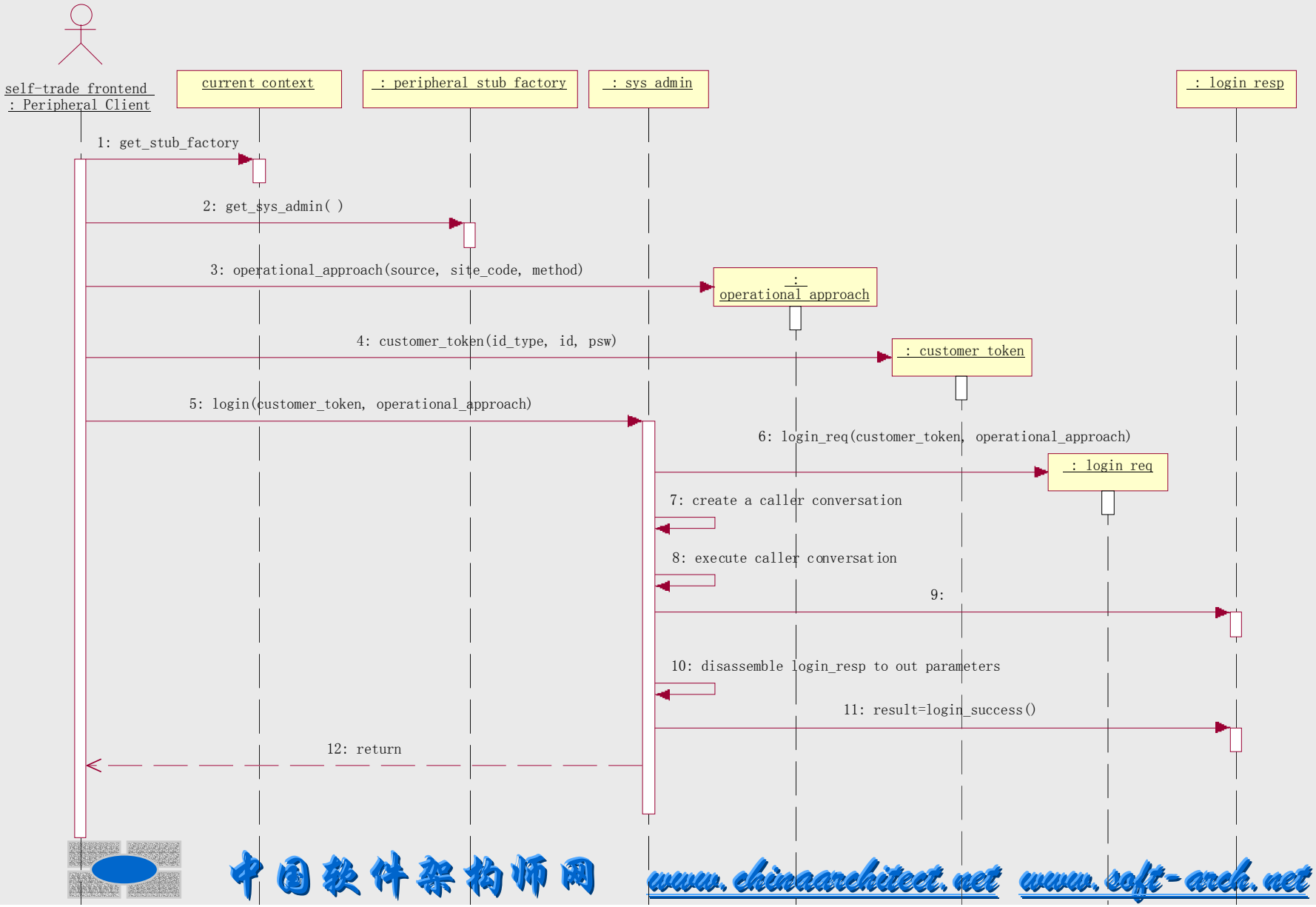
层次结构模型示例


业务服务协议参与者层，包括：依照协议定义调用协议API的服务请求者，和遵循协议定义支持协议API实现的服务提供者
<<layer>>
Business Services Participator
通过Conversation Mechanism的接口间接使用UCS Implementation
<<import>>
<<access>>
可以直接调用Conversation Mechanism完成对话
<<layer>>
Business Services Protocol
<<access>>
Context Components
<<import>>
<<interface>>
Conversation Mechanism
通道平台应用上下文外部构件，包括：外部的协议接口、系统与子系统
<<import>>
通道平台实现层，是对话机制接口层的具体实现
<<layer>>
UCS Platform
被目标系统调用的基础服务和支持构件，通常作为一个整体提供相关领域的某类服务与支持，包括：平台、系统与子系统产品，应用框架等
<<servicePackage>>
Common Utilities
<<servicePackage>>
Infrastructure Components
<<import>>
<<import>>
目标系统私有的公用构件，提供目标系统范围内具有共性的功能，当前不能被其它系统复用，但有进化成为通用底层开发包的潜力
<<servicePackage>>
Foundation Libraries
被直接用来构建目标系统的通用底层开发包，提供各种应用软件较为共性的功能，将被紧密地嵌入目标应用，而且通常没有必须作为一个整体来使用的限制

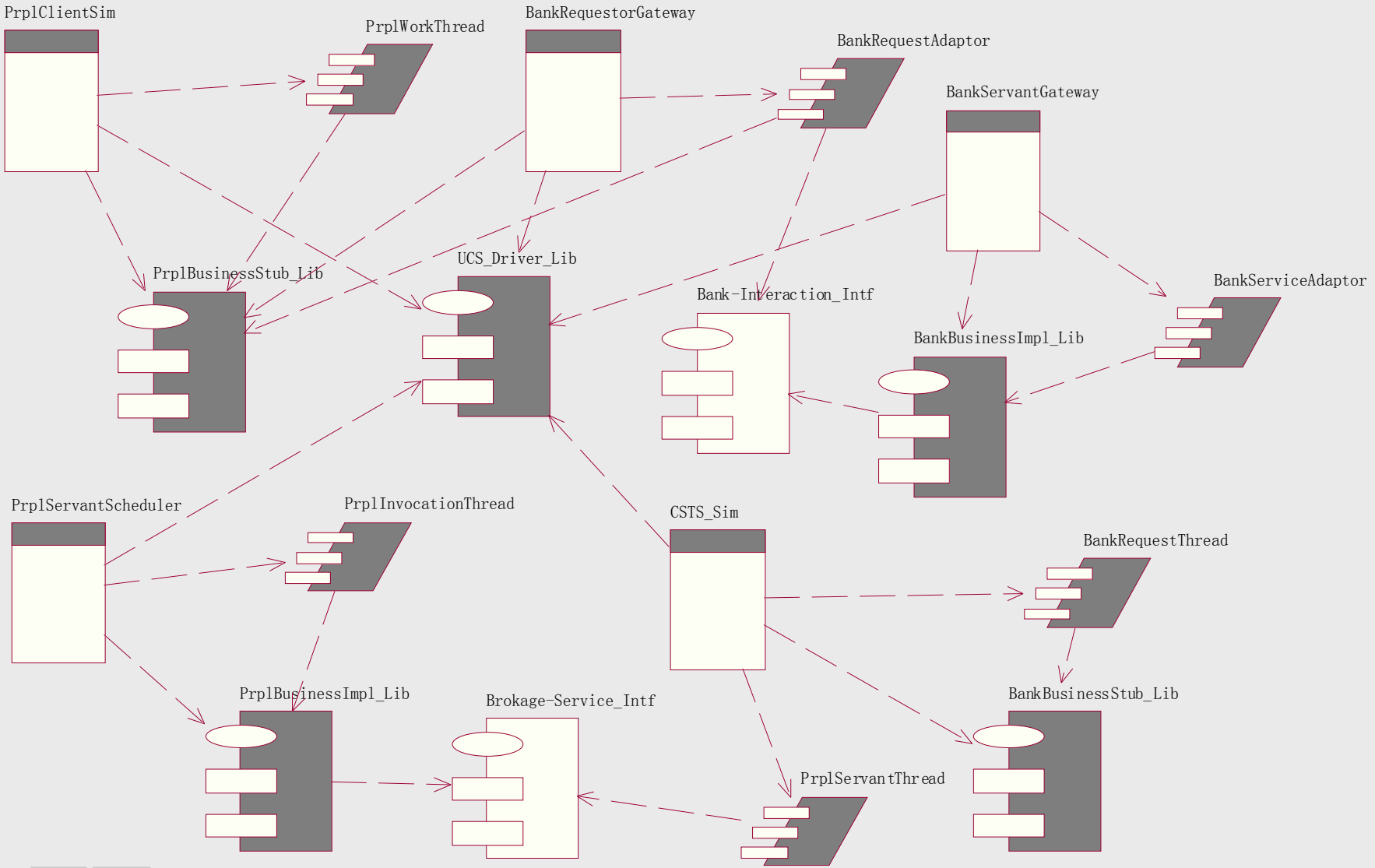

<<singleton>>
stub_factory_base
(from Stub Base)
_service_name
_servant_id
_connection
set_connection()
current_connection()
servant_id()
<<singleton>>
peripheral_stub_factory
get_sys_admin()
get_order_queryor()
get_order_broker()
svrobj_stub_base
(from Stub Base)
_connection
svrobj_stub_base()
sys_admin_stub
sys_admin
(from Core Peripheral Service Obje...)
login()
logout()
order_queryor_stub
order_queryor
(from Core Peripheral Service Obje...)
query_order()
query_order()
order_broker_stub
order_broker
(from Core Peripheral Service Objects)
buy_stock()
sell_stock()

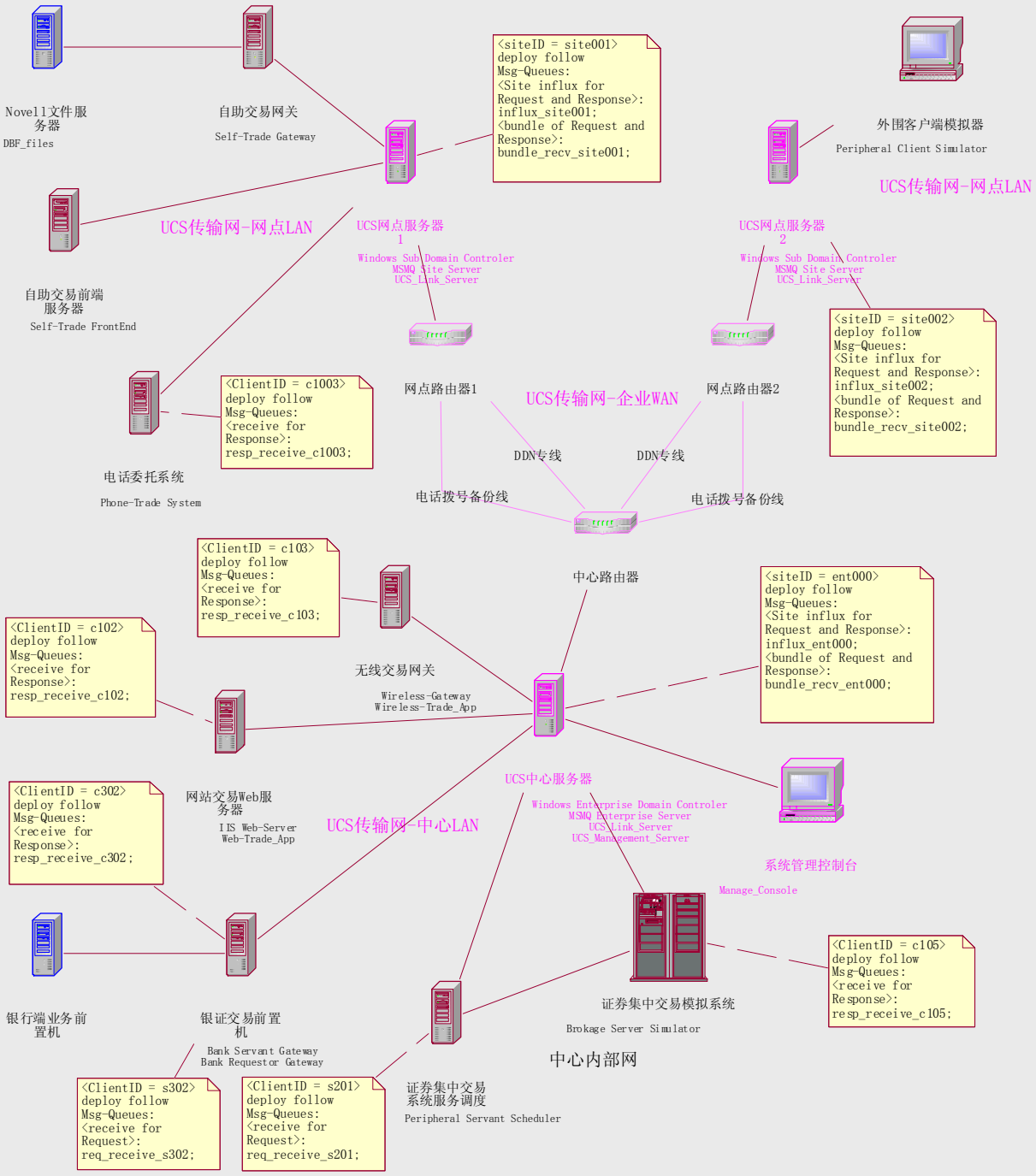
详细设计模型示例

用例实现模型示例



实施（构件）模型示例




Novell文件服务器
DBF_files
自助交易网关
Self-Trade Gateway
<siteID = site001>
deploy follow
Msg-Queues:
<Site influx for Request and Response>:
influx_site001;
<bundle of Request and Response>:
bundle_recv_site001;
外围客户端模拟器
Peripheral Client Simulator
UCS传输网-网点LAN
UCS传输网-网点LAN
UCS网点服务器1
Windows Sub Domain Controler
MSMQ Site Server
UCS_Link_Server
UCS网点服务器2
Windows Sub Domain Controler
MSMQ Site Server
UCS_Link_Server
自助交易前端服务器
Self-Trade FrontEnd
<siteID = site002>
deploy follow
Msg-Queues:
<Site influx for Request and Response>:
influx_site002;
<bundle of Request and Response>:
bundle_recv_site002;
<ClientID = c1003>
deploy follow
Msg-Queues:
<receive for Response>:
resp_receive_c1003;
网点路由器1
UCS传输网-企业WAN
网点路由器2
DDN专线
DDN专线
电话委托系统
Phone-Trade System
电话拨号备份线
电话拨号备份线
<ClientID = c103>
deploy follow
Msg-Queues:
<receive for Response>:
resp_receive_c103;
中心路由器
<siteID = ent000>
deploy follow
Msg-Queues:
<Site influx for Request and Response>:
influx_ent000;
<bundle of Request and Response>:
bundle_recv_ent000;
<ClientID = c102>
deploy follow
Msg-Queues:
<receive for Response>:
resp_receive_c102;
无线交易网关
Wireless-Gateway
Wireless-Trade_App
<ClientID = c302>
deploy follow
Msg-Queues:
<receive for Response>:
resp_receive_c302;
网站交易Web服务器
IIS Web-Server
Web-Trade_App
UCS传输网-中心LAN
UCS中心服务器
Windows Enterprise Domain Controler
MSMQ Enterprise Server
UCS_Link_Server
UCS_Management_Server
系统管理控制台
Manage_Console
<ClientID = c105>
deploy follow
Msg-Queues:
<receive for Response>:
resp_receive_c105;
证券集中交易模拟系统
Brokage Server Simulator
银行端业务前置机
银证交易前置机
Bank Servant Gateway
Bank Requestor Gateway
中心内部网
<ClientID = s302>
deploy follow
Msg-Queues:
<receive for Request>:
req_receive_s302;
<ClientID = s201>
deploy follow
Msg-Queues:
<receive for Request>:
req_receive_s201;
证券集中交易系统服务调度
Peripheral Servant Scheduler

部署模型示例

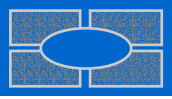
前景文档与目标系统

用前景文档定义目标系统

- 根据软件应用的上下文（或业务建模），将要解决的领域问题，涉众（特别是最终用户）对产品的要求，相关的限制条件等，确定目标系统的定义
- 确定系统的范围，用特性来定义系统，并给出相关的优先级顺序
- 功能性特性将映射到系统用例，并被细化；细化的非功能需求在补充规约中阐明

产品定位

- 确定目标系统的市场背景
- 列明系统将要解决的重大问题
- 系统的概括定义



涉众/用户及其需要

- 标识目标系统的最终用户与其他涉众，以确定需求收集的来源
- 分析用户与涉众的基本特点，以帮助获取与辨别系统的需求
- 列明用户与涉众针对目标系统的各类需要（needs），它们决定了最终系统需求

产品概述

- 明确地定义目标系统
- 勾画目标系统的上下文环境与边界
- 列明目标系统的主要（能力）特性及其提供给客户的利益
- 明示目标系统当前所做的假定和其依赖的条件，它们将可能是未来引起需求变更的重要因素

产品特性

- 以特性（Feature）的方式定义目标系统的高层需求
- 特性表达了目标系统为了实现用户利益而必须具备的能力（Capability）
- 特性是一种对外的服务，通常要求用户提供一系列输入以得到响应的结果

其它高层需求

- 设计约束限定了目标系统设计乃至实现方案的选择范围
- 接口需求
- 质量范围概略描绘了目标系统的重要质量需求
- 适用标准、硬件需求及环境需求等

优先级

- 目标系统的产品特性的重要性优先级大致可以分为：关键、重要和有用
- 特性优先级为项目开发顺序的选择提供了原始依据（当然目标系统各构件间的依赖关系对开发顺序的影响更大，另外开发风险、团队成员的技能和熟悉程度也是重要的评判依据）
- 特性优先级是用于实施需求管理的重要内容

用例规约与补充规约

- 用例详述是对功能性特性的细化
- 系统用例的整体结构适于用UML的Use Case模型表达
- 在UML模型中使用活动图/序列图描述所有的系统用例规格是一种有效的表达方式
- 补充规约是对公共功能性需求和质量等非功能性需求的细化

需求管理

- 需求标识——确定所需要管理的需求类型及其属性与可跟踪性；
- 需求跟踪——建立和管理每一项需求的属性，以确保计划、产品、活动与需求的一致性；
- 需求变更控制——当需求变更时保证交付（产品）的质量和一致性；
- 度量与报告——提供需求相关的可视性（例如需求的实施进度等）；

测试用例

定义——对一系列测试输入、执行状况和预期结果的（通常是正式的）规格说明，用于对目标测试项目的某些特定方面进行评估。

根据意图划分为两类:

用于证实软件需求被实现，通常称为正面测试用例positive test case;

用于证实软件需求只能在期望的条件下被满足，通常称为负面测试用例negative test case，它反映了软件可能面对的无法接受、反常或意外的条件和数据。

测试用例的来源

软件需求无疑是派生测试用例重要来源，但许多情形下这还不够，项目风险、有关约束、特定技术、变更请求（发现的缺陷或错误）等都可能是开发测试用例的参考依据。

派生测试用例的主要途径有——

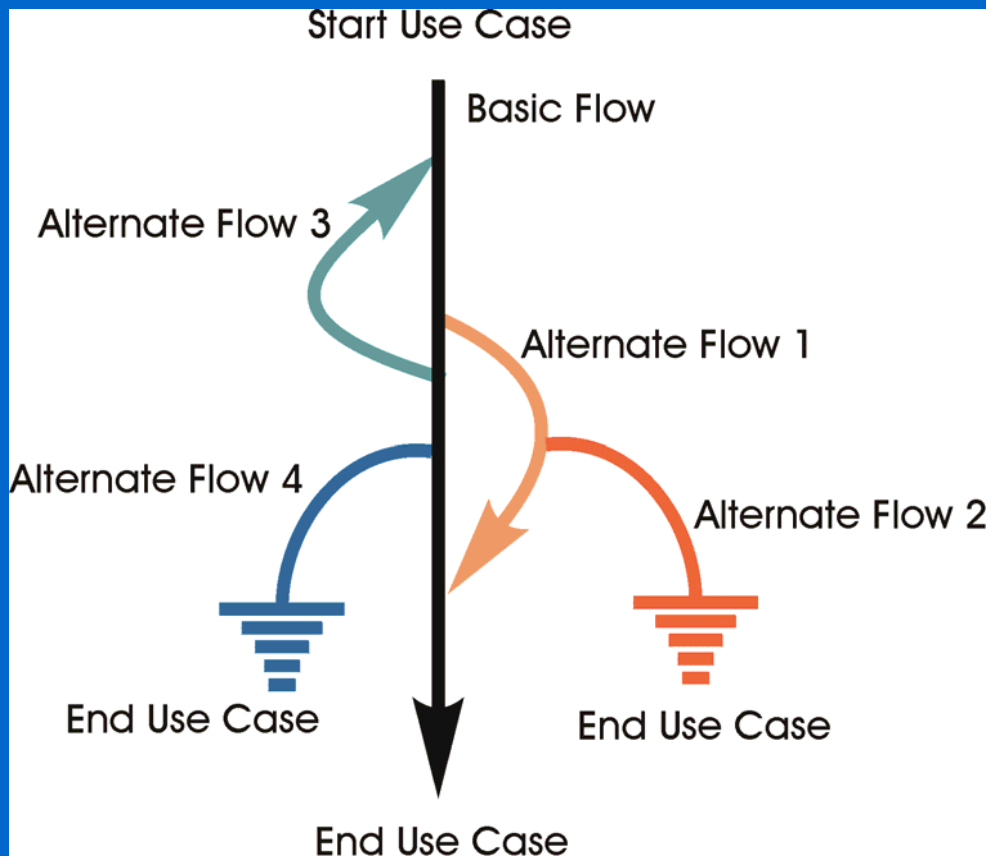
从用例派生功能测试用例；

从补充规约派生非功能测试用例，例如性能测试、安全/访问测试、配置测试、安装测试及其它非功能测试。

从用例派生测试用例

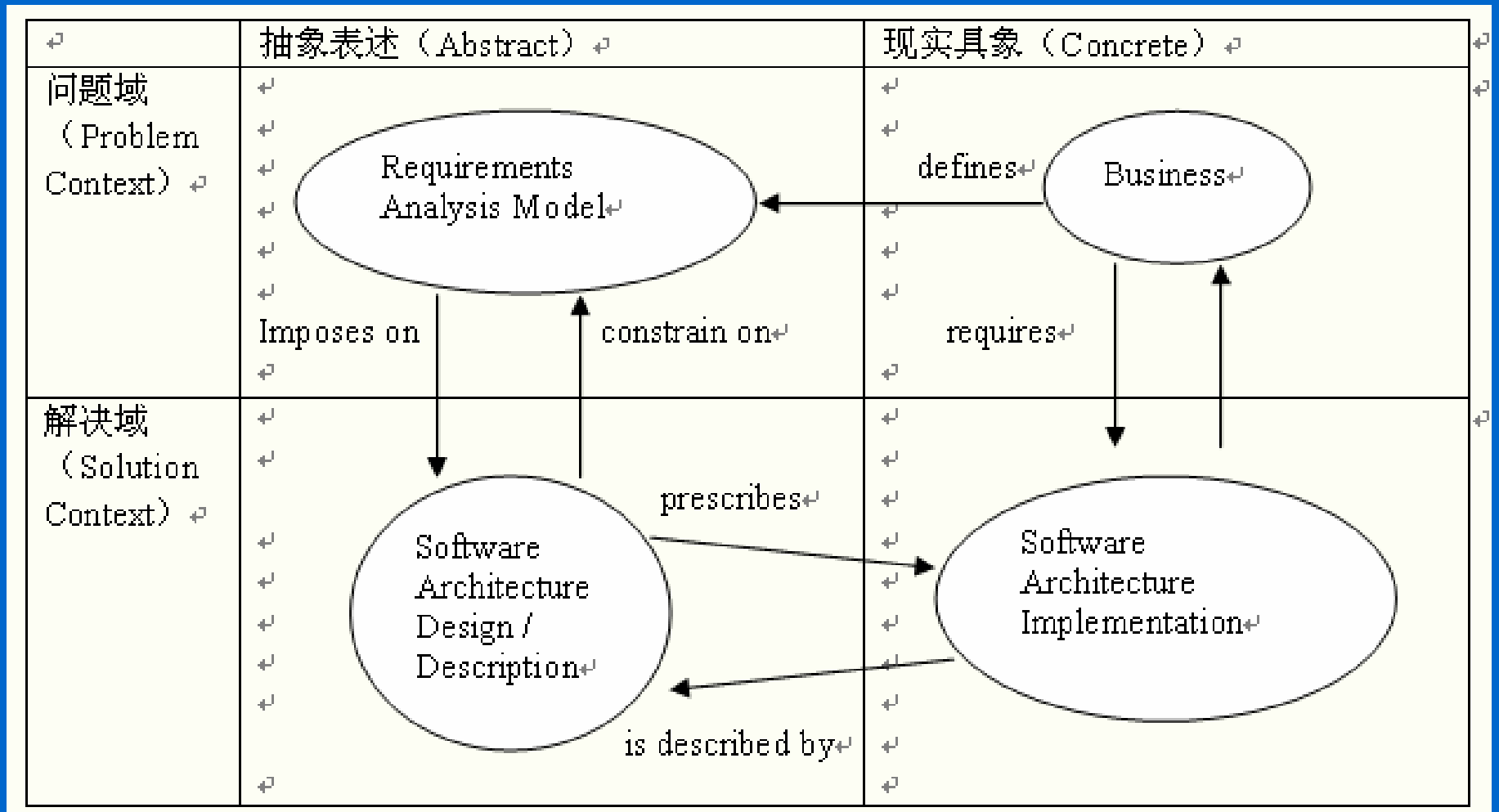
理论上，应当为用例的每个情节scenario开发测试用例；用例情节通过描述用例的执行路径来标识，它从用例开始，贯穿基本流和扩展流直到结束；

图示中用例的情节包括1个基本流与4个扩展流的所有可能的排列组合。



软件构架文档与4+1视图

软件构架表述



构架描述（构架文档）的用途

- 表达（软件）系统及其演化
- 用于系统涉众之间的交流
- 以一致的方式来评估与比较软件构架
- 用来计划、管理与执行系统开发的各项活动
- 表达系统的固有特性与支撑原则，以引导可接受的变更
- 验证系统的实现符合构架描述
- 充实软件密集系统的构架知识库（参考构架）

4+1视图

- 软件系统本身包含的内容太丰富且复杂，就像建筑一样，人们无法同时从一个角度看到其全貌，因此需要使用多个视图（View）来表达系统的构架
- 视图是视点（Viewpoint）的实例，并拥有一个或多个模型（Model）
- 4+1视图分别从外部功能，静态结构，动态行为，运行时刻形态和物理部署拓扑等方面来描述目标软件的构架

构架设计目标与约束

- 构架设计的目标首先要满足目标系统的关键功能需求
- 目标系统的质量需求、接口要求等对构架往往产生决定性的影响
- 项目开发策略，例如第三方构件的选用等是展开构架设计的重要基础
- 变更案例（Change Case）要求构架必须具备相应的可扩展性和适应性
- 设计约束等则限定了构架方案选择的范围

用例视图

- 用例视图从用户使用的角度描述系统构架的基本外部行为特性，通常包含业务用例模型与系统用例模型。
- 通常应选取用例模型中对系统构架的内容产生重大影响的应用场景与用例集合，这些用例代表了系统主要的核心功能，往往决定了系统构架的基本组成元素。

系统概念模型

- 描述目标系统的关键构架机制与概念，主要表达系统为了满足主要软件需求，而采用的相关构架模式、以及引用的重要概念
- 标准的RUP构架文档模板中没有这一部分，但是为了方便涉众理解构架，可以增设此节

逻辑视图

- 逻辑视图从系统内在逻辑结构的角度描述系统的基本结构与动态行为，通常包括分析模型（Analysis Model）、设计模型（Design Model）以及数据模型（Data Model）等。
- 设计模型说明了系统的组成元素、组织架构和关系，并描述了各组成元素的协作以及状态转换关系等（通过用例实现Use Case Realization予以表达）。

进程视图

- 进程视图从系统运行时刻的角度，描述系统划分为进程、线程的结构，及其动态关系。
- 模型主要说明进程、线程的分类，系统构架敏感的主要边界类、控制类对象等在进程、线程中的分布，以及它们之间的创建、交互与消息通讯关系等

部署视图

- 部署视图从系统软硬件物理配置的角度，描述系统的网络逻辑拓扑结构。
- 模型包括各个物理节点的硬件与软件配置，网络的逻辑拓扑结构，节点间的交互与通讯关系等。同时还表达了进程视图中的各个进程具体分配到物理节点的映射关系

实施视图

- 实施视图从软件编译与构建的角度，描述系统实施构件的组织结构与依赖关系（主要是编译依赖）。
- 模型包括实施子系统和构件结构，及其依赖关系。同时还表达了逻辑视图中各个包和类分配到实施视图中的子系统和构件的映射关系

软件质量保证

软件构架文档等作为项目的关键工件，需要得到有效的质量保障，通常采用评审的方式来实行，这些活动属于质量保证的范畴；

SQA过程的目标有——

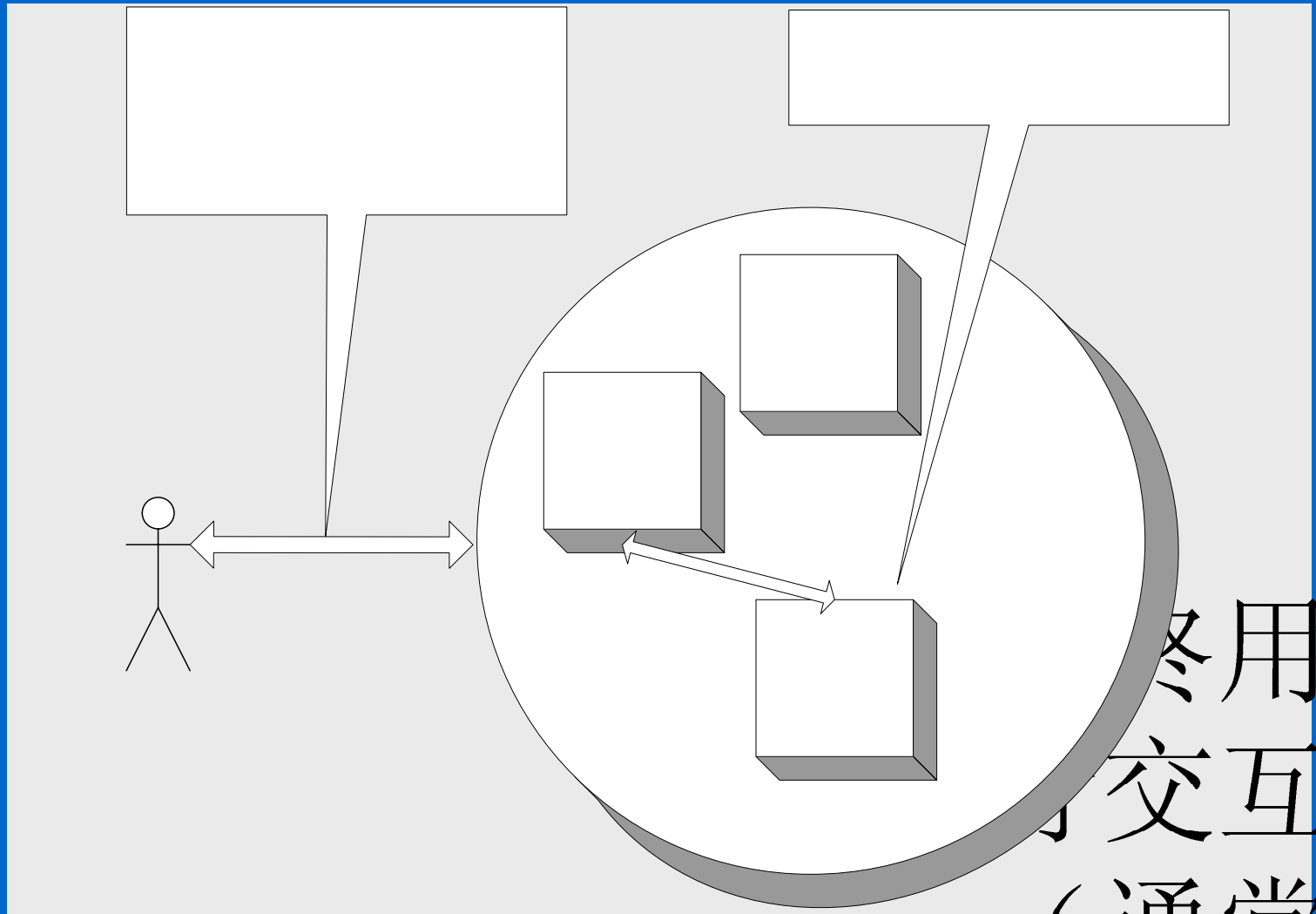
- 软件质量保证活动是有计划的
- 软件开发活动和产出工件是否遵守适用的标准、规程和需求，相关情况必须得到客观的验证
- 受影响的成员应当接到软件质量保证活动和结果的通知
- 分歧、冲突等在项目内部不能解决的问题，应上报高级管理者处理

契约式开发与单元测试

契约无所不在

- 软件系统的本质特征是由表及里、至顶而下的一种层次结构，其构成是所有相对独立的构件或元素
- 而将所有构件或元素组织成为一个有机整体的正是无所不在的契约
- 在最表层，即系统与外部环境之间，是最终用户与系统整体进行交互的契约（通常可以抽象成系统用例）；次之，各子系统/构件之间，是它们相互通讯协作的契约；最后，类与类、类操作、以及独立函数之间，是它们相互调用的契约

软件系统契约关系



终用户与系
交互的契约
（通常可以抽
用例）

契约的内容

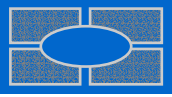
- 前置条件——指在执行某种操作（例如启动用例的一条执行路径、调用类的一个方法等）之前，目标和其上下文必须共同满足的条件（例如系统处于正常运行状态、用户账号存在并未冻结、对象从其上下文获取的资源处于可用状态等）
- 合法的输入——例如发送给子系统的请求消息格式正确、取值在规定范围内等）
- 期望的输出——例如系统返回给用户需要的查询结果等）
- 后置条件——指无论某种操作的执行过程怎样，结束后，目标和其上下文必须达到的状态或满足的条件（例如类的不变式不被打破、上下文资源被释放、系统不出现不能预料的边际效应等）

契约式需求分析（Analysis for contract）

- 如果以契约的观点来观察系统的外部行为，我们不妨在契约式设计之上再引入契约式需求分析的概念，这样契约式开发显得更为完整
- 用例本质上是表述目标系统与其最终用户之间交互的契约，它使得需求规格的定义变得更加精确和全面，而传统的功能及质量需求规格说明往往容易遗漏前置条件、输入格式等重要细节
- 用例强调实现体现了用户利益的目标，在较低层面定义契约时，同样可以显式地描述诸如一个类操作的目标

契约式设计（Design by contract）

- 面向对象的分析、设计其首要问题就是类的划分与职责分配
- 类的职责被确定后，通过定义类的不变式，限定对象的有效状态空间；通过定义类操作的前置、后置条件和输入输出，精确地描述类的行为
- 契约同样有助于精确地表达对象间的协作，因为这些协作步骤将遵从一系列的契约
- 契约式设计可以通过测试驱动开发等最佳实践来驱使贯彻



契约式编程

- 契约式编程要求客户（client）代码在调用服务（server）代码时也要遵守契约，意味着双方共同承担使运行获得成功的责任，使得代码间职责的分配更为均衡、合理，避免了服务代码中防错式设计的过度蔓延
- 在代码中加入判别契约是否被遵守的语句（例如assert），使得编码中的缺陷（bug）能及时地暴露出来
- 契约式编程结合测试先行、单元测试等最佳实践，是实现高质量构造（construct）的捷径

测试驱动开发

- XP推荐测试先行，即先编写测试代码，之后以通过所有测试为目标来驱动实现源码的开发，这通常也称作测试驱动开发
- 先编写测试代码，使得实施员对目标源码的外部行为能先建立明确无误的理解，并有助于尽早发现设计上的缺陷（赶在实现之前，毕竟编写测试代码的开销较小）
- 测试先行的实质，就是先通过测试代码最精确地表达目标代码的契约（即定义其需求），随后每次测试执行都是为了验证源码是否满足了契约

单元测试

- 单元测试比集成测试能更大限度地覆盖单元的执行路径，是保证软件质量性价比最高的途径
- 单元测试代码提供了目标代码最直接、便利和相对独立的运行上下文，方便代码的调试和除错
- 单元测试在XUnit等测试框架的支持下，最容易实现测试的自动化，这也是回归测试所依赖的重要基础
- 实施回归测试，增强了项目的可视性，能及时提供反馈，极大地加强了开发的可控性

单元测试编写原则

- 测试任何可能出错的地方，对于明显不太可能出错的方法（譬如set 和get这些非常简单的方法），单元测试几乎没有意义
- 注意测试边界条件，比如未初始化、NULL、最大最小值等，防止实现时忘记处理它们
- 为目标代码编写独立的单元测试代码，尽量不要与其它代码产生依赖
- 针对接口进行测试

测试代码示例

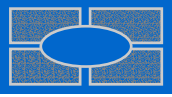
```
CPPUNIT_TEST_SUITE_REGISTRATION(conversation_test);
// 单线程测试caller_conversation
void conversation_test::singlethread_caller_conversation()
{
    typedef service1<mock_executor,mock_manager*> single_service;
    run_notify
     cur_notify("conversation_test::singlethread_caller_conversation");
    mock_manager test_manager;
    single_service svr(&test_manager);
    scope_guard _guard;

     _guard.register_todo(single_service::timed_join,&svr,DEFAULT_WAIT_M
     S);
    performance_time action;
    action.start();
    {
          caller_tester  single;
       for(int i=0 ; i< 1000; ++i)
       {
          single.do_task();
          cur_notify.watch(_WATCH_HELPER(i));
       }
```

自动化构建与持续集成

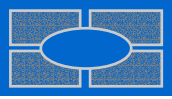
集成与构建

- 软件开发的目标是得到满足需求的可运行的交付工件，即通常是将源码等中间工件编译、链接并集成而生成的一个建造（build）
- 构建集成是一项看似简单实际上充满了陷阱的工作，在团队开发的场景下，将牵涉到将不同成员开发的源码等集成一体，解决各类冲突与依赖等复杂情况，这个过程直接依赖于软件配置管理流程的支持
- 一个合格的集成员需要掌握多项知识和技能



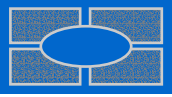
集成与构建的内容

- 配置项目集成的开发工具（列如msvc6/bcc55等编译器）和自动构建工具（列如ant/cpptasks）
- 配置软件配置管理环境（例如clearcase客户端）
- 制定不同构件间的编译引用、库链接等集成原则
- 确定针对第三方开发包的源码结构组织与构建步骤
- 确定针对项目构件的源码结构组织与构建步骤
- 编制构建脚本
- 制定集成计划
- 执行实施—〉单元测试—〉提交—〉集成—〉冒烟测试流程



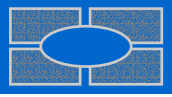
构建计划

- 说明要在此迭代中实施哪些子系统/构件，并说明为及时做好集成准备而实施这些子系统的首选顺序，这一顺序取决于构件间的依赖关系
- 列明增量集成的工作版本（Working Version），每次迭代可能包含多个可测试的集成构造版本，它们决定了每次集成构建周期

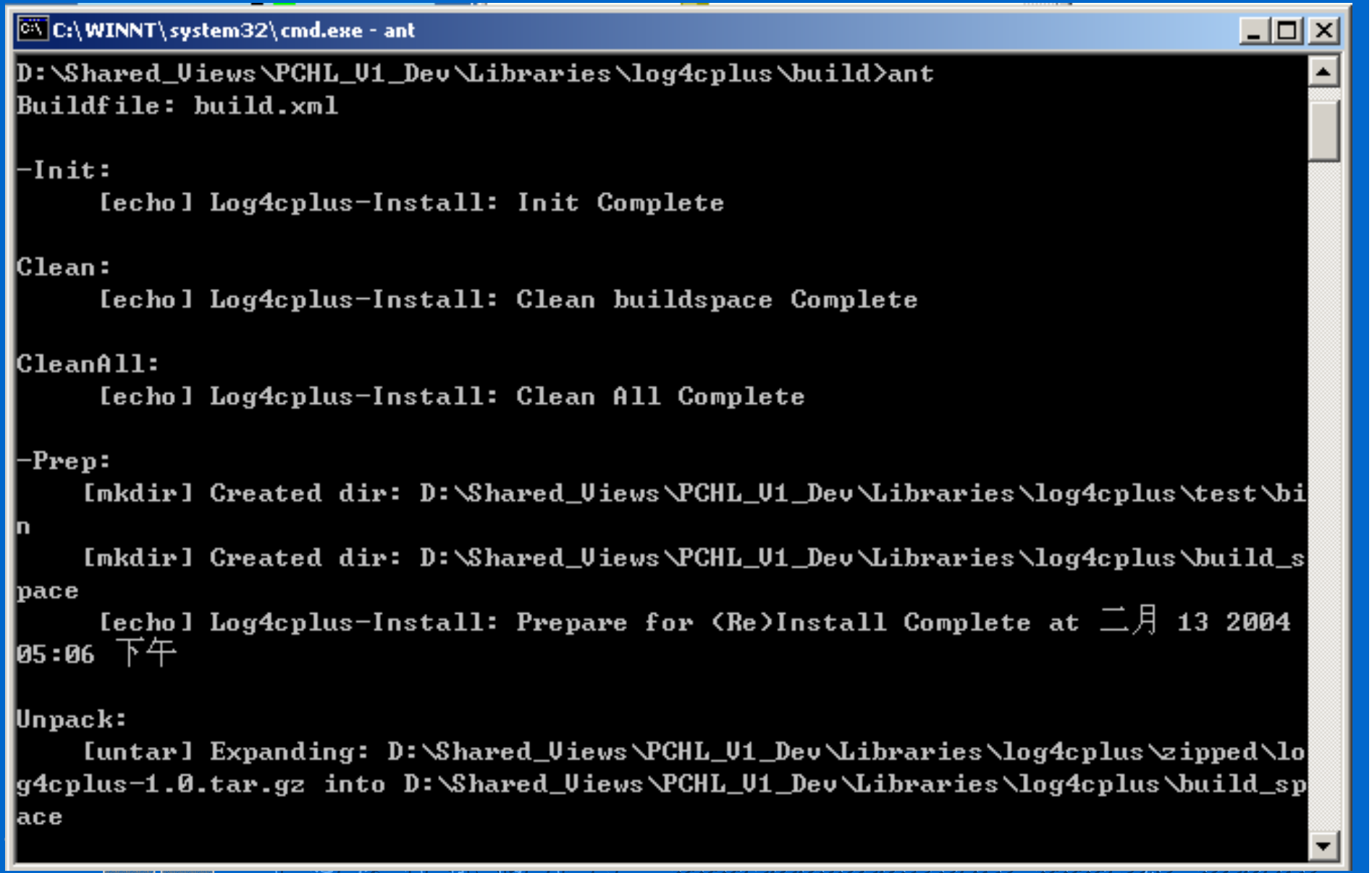


Ant构建脚本

- Ant配置文件描述了一个构建项目（project），它由一些属性定义（property）和一个目标树（target tree）组成;
- 目标代表了一个期望的构建结果（例如生成一个链接好的可执行文件），并表述了其依赖的其它目标，常见的构建目标有初始化（Init）、编译（Compile）、单元测试（Test）、安装（Install）、清除（Clean）等;
- 每个目标包含了实现它而将要执行的任务（task），Ant支持的任务种类非常丰富，例如源码编译、文件拷贝、执行命令行操作等。

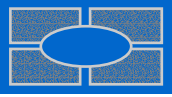


Ant执行示例

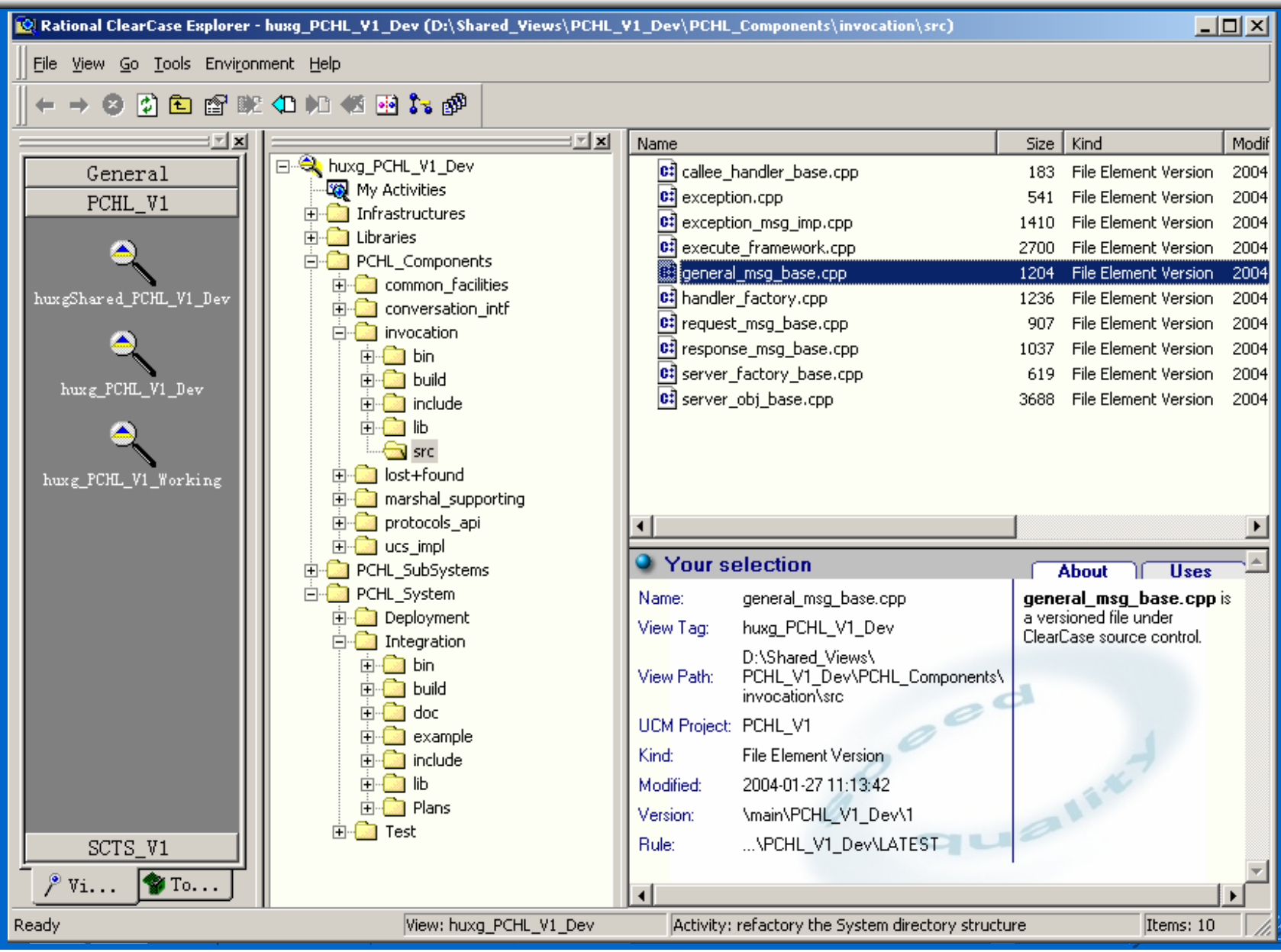

C:\WINNT\system32\cmd.exe - ant
D:\Shared_Views\PCHL_V1_Dev\Libraries\log4cplus\build>ant
Buildfile: build.xml
-Init:
[echo] Log4cplus-Install: Init Complete
Clean:
[echo] Log4cplus-Install: Clean buildspace Complete
CleanAll:
[echo] Log4cplus-Install: Clean All Complete
-Prep:
[mkdir] Created dir: D:\Shared_Views\PCHL_V1_Dev\Libraries\log4cplus\test\bi
n
[mkdir] Created dir: D:\Shared_Views\PCHL_V1_Dev\Libraries\log4cplus\build_s
pace
[echo] Log4cplus-Install: Prepare for (Re)Install Complete at 二月 13 2004
05:06 下午
Unpack:
[untar] Expanding: D:\Shared_Views\PCHL_V1_Dev\Libraries\log4cplus\zipped\lo
g4cplus-1.0.tar.gz into D:\Shared_Views\PCHL_V1_Dev\Libraries\log4cplus\build_sp
ace

源码目录组织原则

- 源码组织结构属于开发（产品）目录的组成部分，提供了项目团队进行编码、调试、构建（编译）与集成等活动的统一共享场所，它需要满足不同角色（实施员、集成员、测试员等），在不同的阶段（编码、单元测试、集成、集成测试、系统测试等），对不同类型工件进行访问的多种场景需求
- 因为项目的编码实施、集成等活动相互间的依赖关系远比其它文档编写类活动要复杂，协同整合更为困难，使得合理的源码组织结构变得极为重要
- 构架师和配置管理员必须投入更多精力关注开发（产品）目录中的源码组织部分

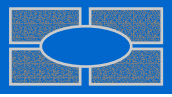


源码目录结构示例

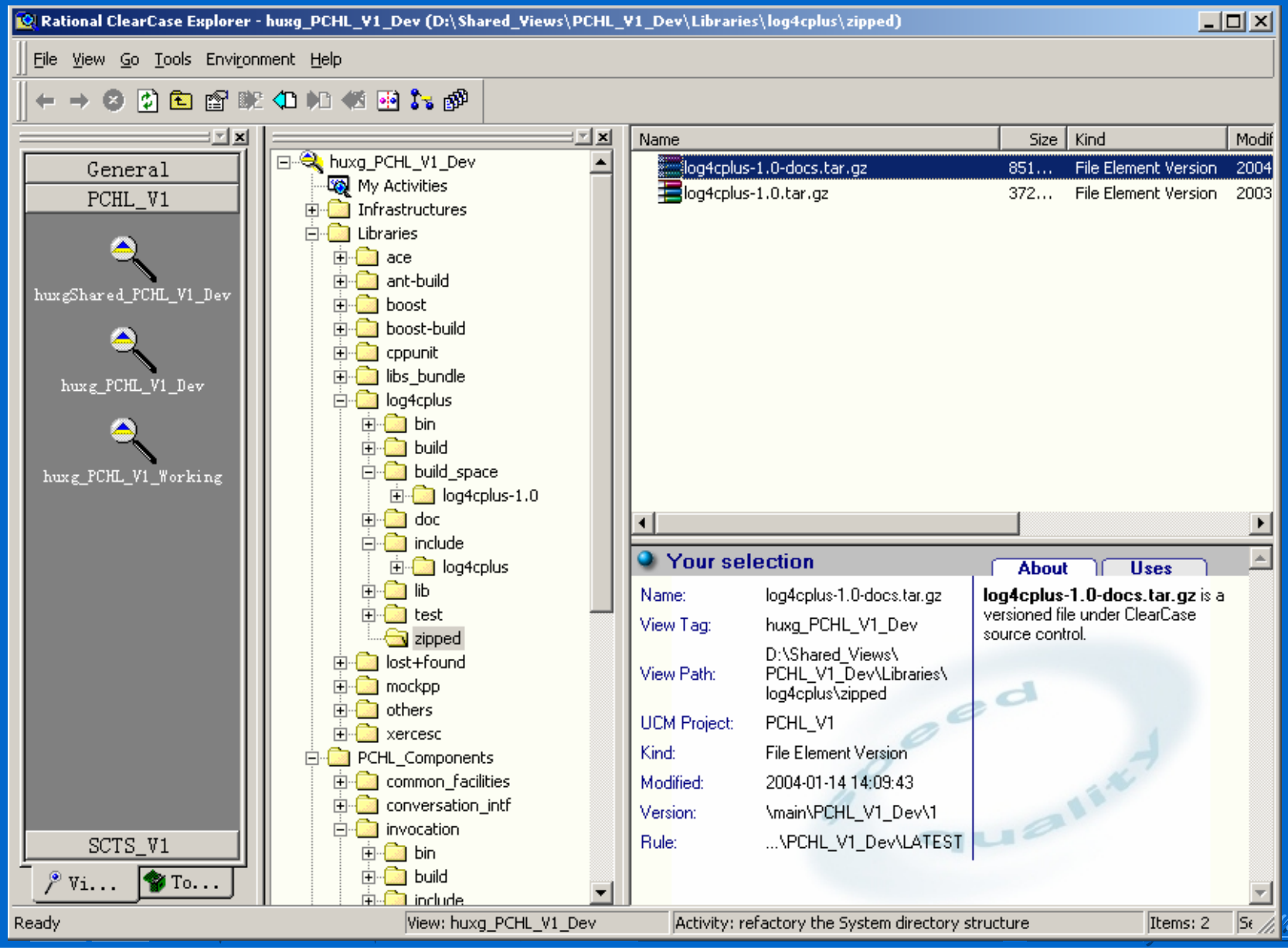

Rational ClearCase Explorer - huxg_PCHL_V1_Dev (D:\Shared_Views\PCHL_V1_Dev\PCHL_Components\invocation\src)
File View Go Tools Environment Help
General
PCHL_V1
huxgShared_PCHL_V1_Dev
huxg_PCHL_V1_Dev
huxg_PCHL_V1_Working
SCTS_V1
Vi...
To...
huxg_PCHL_V1_Dev
My Activities
Infrastructures
Libraries
PCHL_Components
common_facilities
conversation_intf
invocation
bin
build
include
lib
src
lost+found
marshal_supporting
protocols_api
ucs_impl
PCHL_SubSystems
PCHL_System
Deployment
Integration
bin
build
doc
example
include
lib
Plans
Test
Name Size Kind Modif
callee_handler_base.cpp 183 File Element Version 2004
exception.cpp 541 File Element Version 2004
exception_msg_imp.cpp 1410 File Element Version 2004
execute_framework.cpp 2700 File Element Version 2004
general_msg_base.cpp 1204 File Element Version 2004
handler_factory.cpp 1236 File Element Version 2004
request_msg_base.cpp 907 File Element Version 2004
response_msg_base.cpp 1037 File Element Version 2004
server_factory_base.cpp 619 File Element Version 2004
server_obj_base.cpp 3688 File Element Version 2004
Your selection
About
Uses
Name: general_msg_base.cpp
View Tag: huxg_PCHL_V1_Dev
View Path: D:\Shared_Views\PCHL_V1_Dev\PCHL_Components\invocation\src
UCM Project: PCHL_V1
Kind: File Element Version
Modified: 2004-01-27 11:13:42
Version: \main\PCHL_V1_Dev\1
Rule: ...\PCHL_V1_Dev\LATEST
general_msg_base.cpp is a versioned file under ClearCase source control.
Ready
View: huxg_PCHL_V1_Dev
Activity: refactory the System directory structure
Items: 10

第三方开发包的源码结构组织

- 第三方开发包每个产品有各自不同的目录结构，组织的方式不统一，直接使用将增加引用和依赖关系的复杂性；
- 产品目录全部展开后有时文件数量非常庞大，如果直接纳入配置管理的话，加入源码控制的开销很大，而当其版本升级时替换原有文件更是非常繁琐且容易出错，但是不控制的话又会造成第三方开发包版本冲突和安装路径不一致的问题
- 项目中对第三方开发包的引用，通常不直接使用其源码，而是链接其编译好的静态库。



第三方开发包构建方案示例



第三方开发包构建脚本示例

```
<?xml version="1.0"?>
<project name="Log4cplus-Install" default="InstallAll" basedir=".">
<description >
    Build file for all log4cplus library project
</description>
    <!-- task and type extension -->
    <taskdef resource="cpptasks.tasks"/>
    <typedef resource="cpptasks.types"/>
    <taskdef resource="net/sf/antcontrib/antcontrib.properties"/>

    <target name="AutoBuild" depends="-Init" description="AutoBuild all source to get
    bin out">
    <if>
        <equals arg1="${cc}" arg2="msvc"/>
    <then>
        <available file="${src_dsw}" filepath="${int.msvchome}"
    property="MsvcProjExist"/>
        <assert name="MsvcProjExist">
            <apply executable="msdev" dir="${int.msvchome}"
    failonerror="true"
                vmlauncher="false" os="Windows 2000">
                <srcfile />
                <arg value="/MAKE"/>
                <arg value="${proj_list}"/>
                <fileset dir="${int.msvchome}">
                    <include name="${src_dsw}"/>
                </fileset>
            </apply>
        </assert>
        <available file="${unisrc_dsw}" filepath="${int.msvchome}"
```

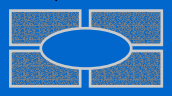
迭代开发模式

软件项目管理

项目管理是一门平衡规模（产品特性）、资源和进度的艺术；其内容包括——

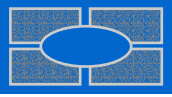
- 项目估计与计划
- 计划执行的跟踪和监督

项目管理应当使得项目：推进有序，并保持稳定的节奏，让所有成员的工作步调一致；最大限度地规避风险，让风险提前暴露，以提供解决风险的足够时间和机会；减少相互依赖，尽量提早进度，同时提高资源的利用效率。



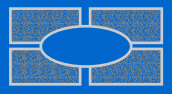
迭代开发模式

- 软件的不确定和高风险等特性，使得传统的瀑布式开发力不从心
- 迭代有助于尽快发现和解决风险
- 迭代有助于控制项目的节奏，加快反馈，增强项目的控制力度，实现过程的有序化
- 迭代符合人们对事物的认识逐步加深，解决问题的能力随经验逐步提高，人类最根本的技能就是实践、总结和学习的能力等客观事实



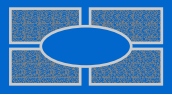
迭代计划

- 确定本次迭代的目标，这包括：技术与开发、方法与过程、团队与管理三个方面
- 定义迭代产出的工件，例如文档、模型、代码等
- 细化迭代的任务



迭代总结

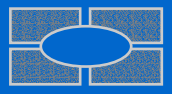
- 回顾上次迭代目标的达成情况
- 回顾迭代进度执行结果
- 统计迭代工件产出量，例如文档页数、代码行等
- 简述迭代集成测试的结果
- 总结经验与教训，例如团队建设、内外沟通、项目开发过程、需求、设计等各方面的成功经验和失败教训
- 提出改进建议



其它内容

- 业务建模
- 系统测试
- 软件配置管理
- 使用优秀的开放源码
- Rational Case工具的使用
- 开发/工作网络环境配置

…



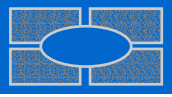
结语

软件开发内在的根本复杂性、不确定（可变）、不可见等，决定了没有一种方法或途径能够解决软件项目中所有的问题，即所谓“没有银弹”。

团队需要在一个统一的环境中开展工作，他们所使用的工件应当保持一致和完整；必须有一种有效的机制来支持成员的协同，避免混乱和无序。

团队协作中的首要问题就是沟通，不同的成员有不同的经历和知识背景，表达技巧和理解能力也参差不齐，如何避免沟通中的信息丢失和理解错误，是关系团队成败的关键因素。

团队需要有一个共同的愿景，以集中成员的注意力，并朝向一个正确的目标努力，最终开发出客户真正期望和需要的产品。



结语

团队中每个成员都对软件系统存在自己的想法和期望，如何集思广益、发挥每个人的长处，同时又能集中和统一为一种思路，从而确保软件构架的概念完整性（一致和完备），这是软件开发永远的核心问题。

有了良好的软件构架，相当于给开发者一个美好的蓝图；然而如何正确地实施它，以将蓝图转变成现实，高质量的软件构建在此扮演了核心的角色。

在个人英雄主义盛行的年代，构建与集成的重要性并没有得到重视，然而在团队协作的环境下，集成成了宣布项目成功或失败的关键一票；独自完成整体的一个部分并不困难，然而让不同的人做的构件集成为一个整体，却不是一件容易的事。

传统的瀑布开发模式已经适应不了软件管理的现实要求，需求总是在变化，项目风险总是隐藏在深处，团队的经验需要给予更多的机会来实践总结，这一切决定了迭代开发模式的诞生。

Q&A

谢谢！