IBM Software Group

如何能够在系统开发领域取得领先优势？

IBM Rational系统开发解决方案

王家欣
北方区经理
wangjiax@cn.ibm.com

Rational. software

ON DEMAND BUSINESS

一些需要考虑的问题

- 您是带着什么样的期望来参加本次活动的？
- 您所在单位当前在嵌入式系统开发方面的具体状况是什么样？
- 您在嵌入式系统开发方面遇到了什么样的问题？
- 这些问题对于您所在单位的发展有影响吗？如果有的话，是什么影响？
- 您所在单位有意愿来解决这些问题吗？有解决方案吗？
- 您所在单位在解决开发中的问题的过程中遇到问题吗？是什么？
- 您认为这些问题的根源是什么？

IBM Rational要为大家做什么？

- IBM的价值观：
  - Customer success
  - Trust
  - Innovation
- Rational的使命——帮助客户解决发展中面临的难题，帮助客户开创“蓝海”
  - IBM曾经失去对于嵌入式系统开发领域的关注
  - 我们曾经一度忽视从客户的角度来考虑客户的问题
  - 转变就从现在开始
- 我们努力从客户的价值观角度考虑客户的问题
  - 从客户的需求出发——产品体系快速发展、有效整合
  - 为客户提供解决方案——“量体裁衣”
  - 确保客户成功——创造价值
- 从一点一滴做起
  - 与您互动——您的积极反馈是对于我们的巨大帮助，同时也是帮您

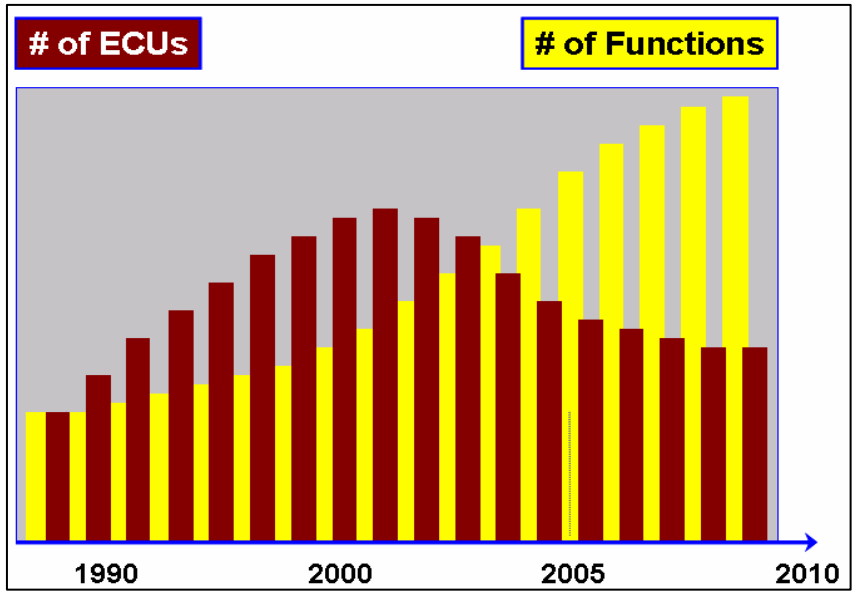
嵌入式系统的发展迅速，应用领域迅速扩展

- 空间技术 & 国防领域
- 汽车 & 汽车电子
- 电子消费品领域
- 工业自动化
- 医疗
- 通讯领域
- 环境工程
- … …

- 有着巨大的发展空间，有更多的创新机遇

功能越来越多，系统组件越来越少

- 系统的组件有越来越少的趋势
- 功能越来越多地集成在系统的软件组件中
- 系统的组件之间需要更为密切的通讯，更为紧密的协作
- 从产品的可扩展性和易于维护的角度考虑，需要组件之间能够实现松散的耦合，以便容易替换和重用

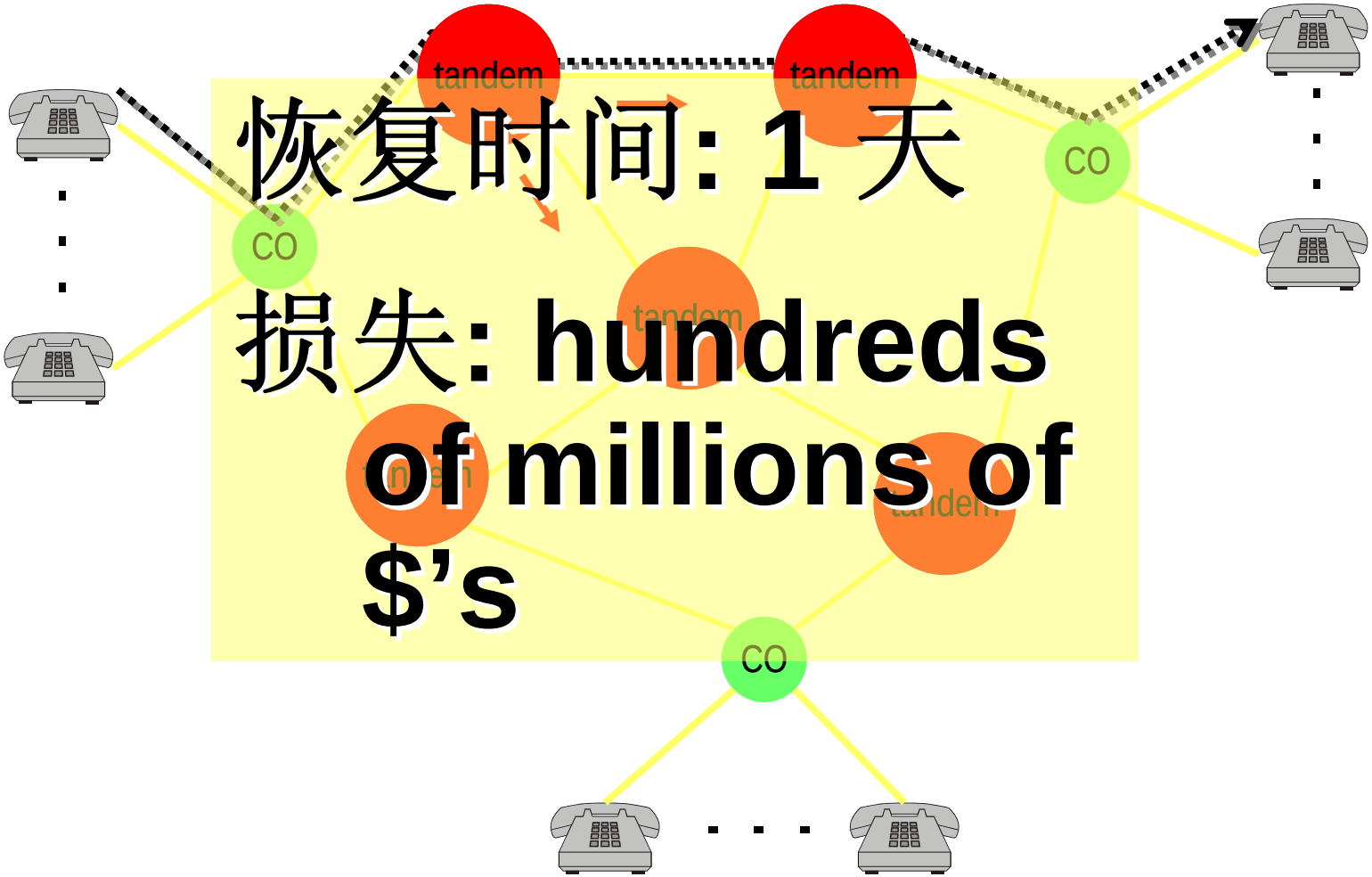


越来越多的嵌入式软件，越来越复杂的嵌入式软件

- 嵌入式系统的开发向着大型化、复杂化的方向发展

一个重大的工程化的灾难

- 1990: AT&T 远距离的传输网络 (美国东北部)



问题的根本原因

- 在一个软件模块中缺少了 “break” 语句
  - 在数百万行(X,000,000)的代码中 缺失了(1)1行代码

```
. . .;
switch (...) {
      case a : ...;
            break;
      case b :...;
            break;
            . . .
      case m : ...;
      case n : ...;
            . . .
};
```

认为执行到这里就应该结束了，没有注意到会执行进入下一个”case”中去

Q: 为什么写出高质量的软件如此困难?

A: 复杂性!

当今的软件复杂度已经达到了接近生物系统的程度；有些时候一个系统由多个系统所组成，并且每个系统都包含上千万行的代码

...每一个系统出现问题都会给整个的系统巨大的损失

Fred Brooks 对于复杂性的分析

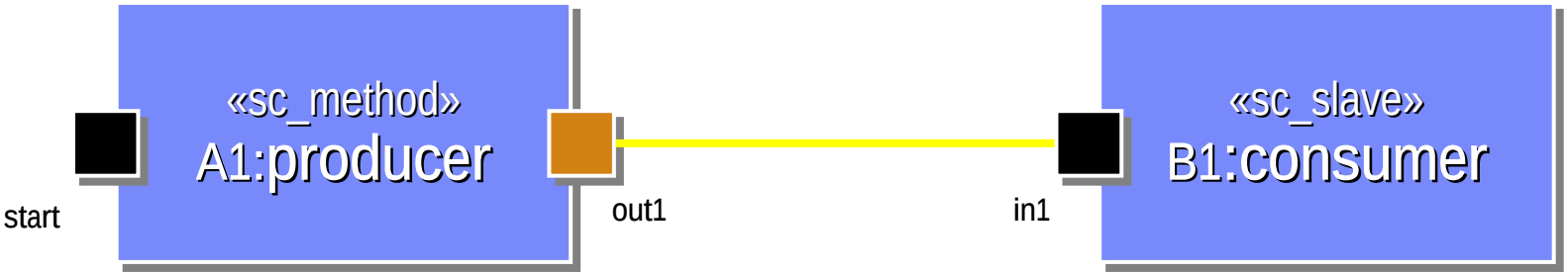
- [From: F. Brooks, "*The Mythical Man-Month*", Addison Wesley, 1995]
- *Essential 本质的复杂性*
  - 问题本身所固有的
  - 技术和技巧都无法消除的
  - 例如：应对并发行的问题
- *Accidental 非本质的复杂性*
  - 由我们处理问题的方法和技术所导致的
  - 例如：建造摩天大楼却只使用手工工具

看一个实例...

```
SC_MODULE(producer)
{
sc_outmaster<int> out1;
sc_in<bool> start; // kick-start
void generate_data ()
{
for(int i =0; i <10; i++) {
out1 =i ; //to invoke slave;}
}
SC_CTOR(producer)
{
SC_METHOD(generate_data);
sensitive << start;}};
SC_MODULE(consumer)
{
sc_inslave<int> in1;
int sum; // state variable
void accumulate (){
sum += in1;
cout << "Sum = " << sum << endl;}
SC_CTOR(consumer)
{
SC_SLAVE(accumulate, in1);
sum = 0; // initialize
};
SC_MODULE(top) // container
{
producer *A1;
consumer *B1;
sc_link_mp<int> link1;
SC_CTOR(top)
{
A1 = new producer("A1");
A1.out1(link1);
B1 = new consumer("B1");
B1.in1(link1);}};
```

你能够看出来这个软件做什么吗？

这个软件的模型

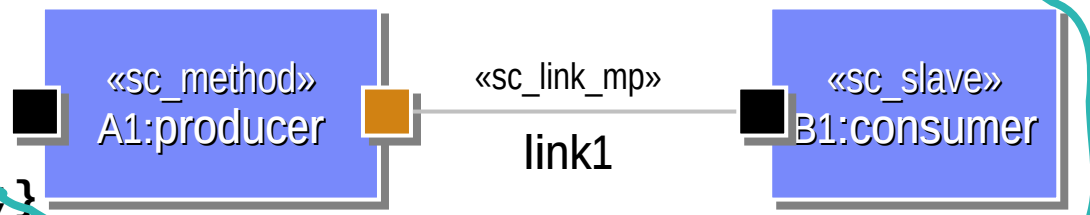


现在能够看出来吗?

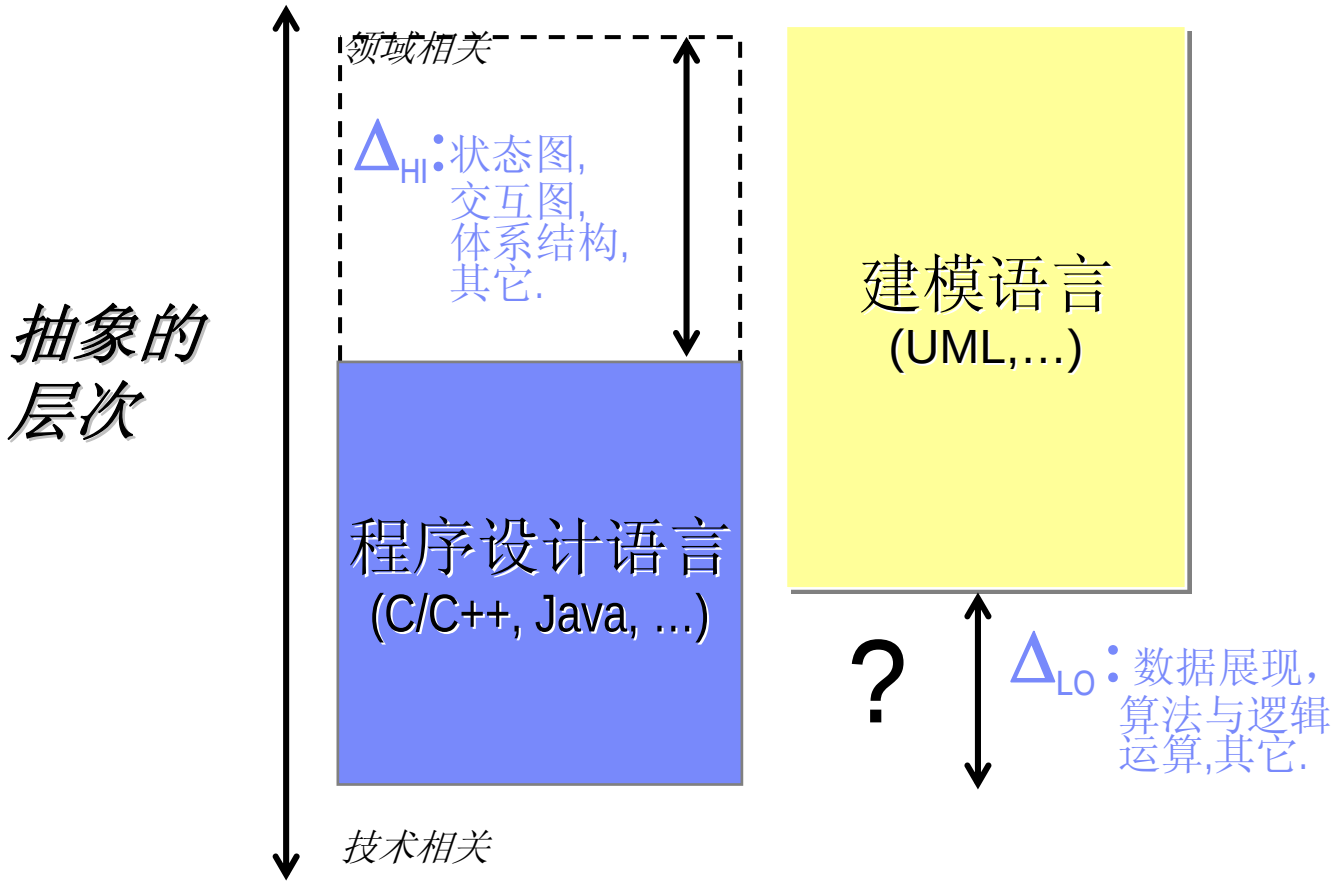
模型与代码

```
SC_MODULE(producer)
{
sc_outmaster<int> out1;
sc_in<bool> start; // kick-start
void generate_data ()
{
for(int i =0; i <10; i++) {
out1 =i ; //to invoke slave;}
}
SC_CTOR(producer)
{
SC_METHOD(generate_data);
sensitive << start;}};
SC_MODULE(consumer)
{
sc_inslave<int> in1;
int sum; // state variable
void accumulate (){
sum += in1;
cout << “Sum = “ << sum << endl;}
```

```
SC_CTOR(consumer)
{
SC_SLAVE(accumulate, in1);
sum = 0; // initialize
};
SC_MODULE(top) // container
{
producer *A1;
consumer *B1;
sc_link_mp<int> link1;
SC_CTOR(top)
{
A1 = new producer(“A1”);
A1.out1(link1);
B1 = new consumer(“B1”);
B1.in1(link1);}};
```



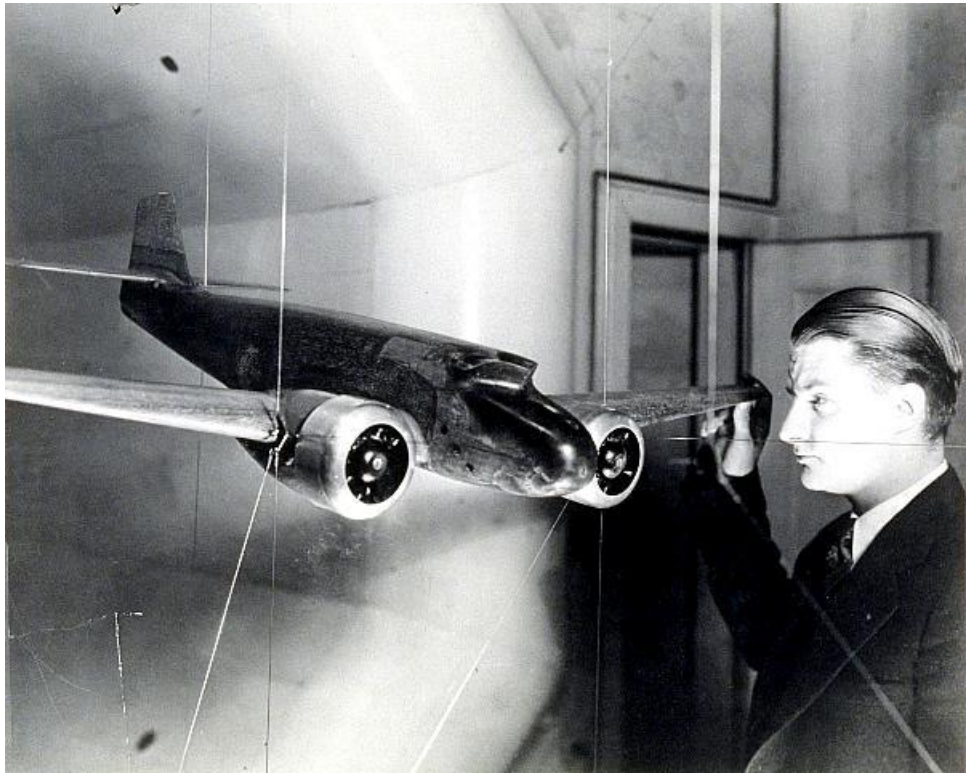
建模语言与程序设计语言



模型在工程化中的使用

- 可能和工程化本身的历史一样长 (c.f., Vitruvius)
- 工程化模型:

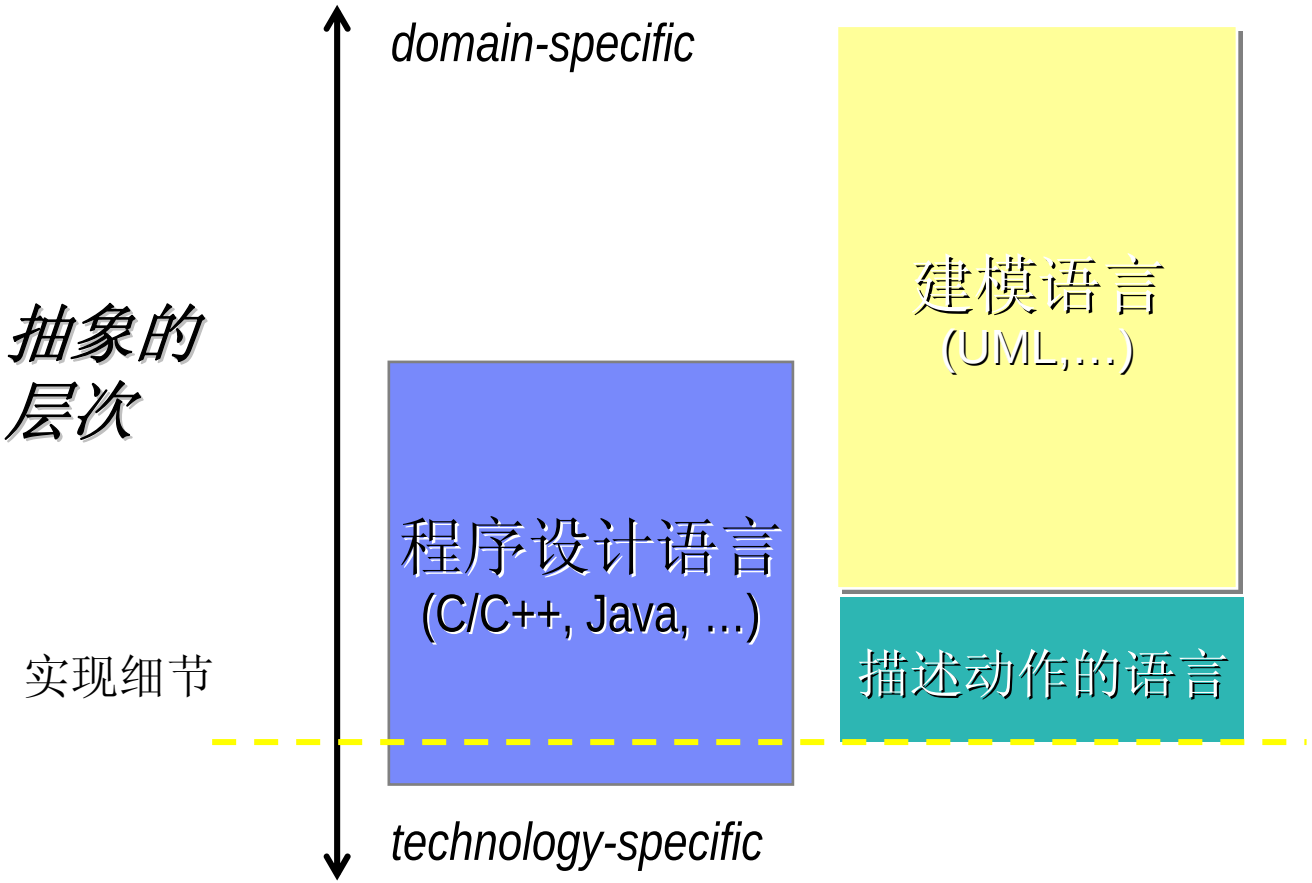
  一个对于系统的一个简化了的展现，突出了那些从一个给定的视点观察到的感兴趣的特征



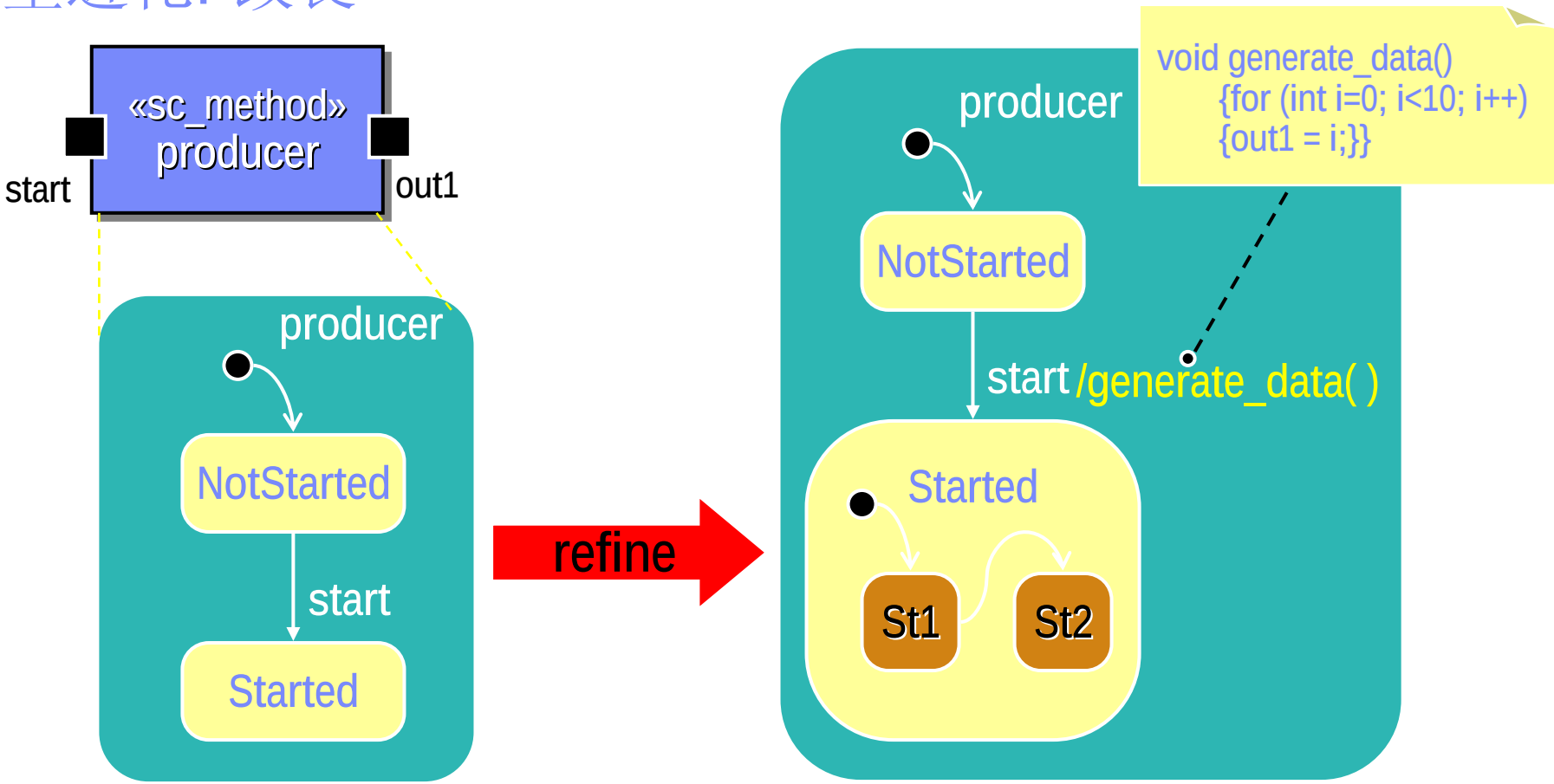
- 我们不能够在一次观察看到所有的东西
- 我们所见到的针对人们的理解需要而调整了的

软件建模呢?

软件模型: 填补必要的细节



模型进化: 改良



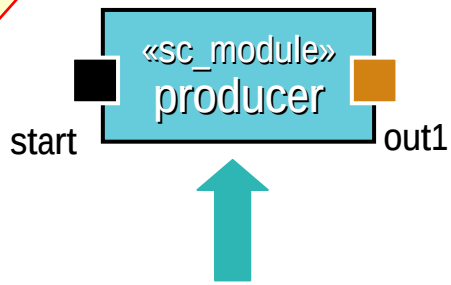
- 模型能够被不断地改良指导应用被完整地描述出来 ⇒ *模型也由此变成了你正在建模的系统!*
- *软件是一个唯一的通过对于模型的改良最后能够直接获得最终产品的*

模型驱动的开发 (MDD)

- 一个软件开发的方法，在这个方法中开发所关注的并且是最关键的工件就是模型（不同于程序）
- 基于两个经过长期验证了的方法:

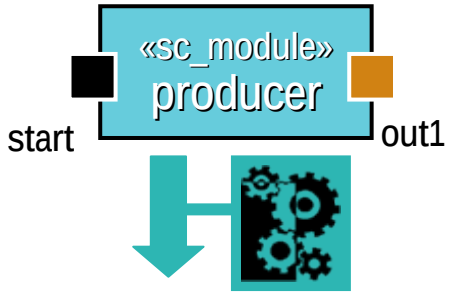
(1) ABSTRACTION

关系到
建模语言




```
SC_MODULE(producer)
{sc_inslave<int> in1;
int sum; //
void accumulate (){
sum += in1;
cout << "Sum = " <<
sum << endl;}
```

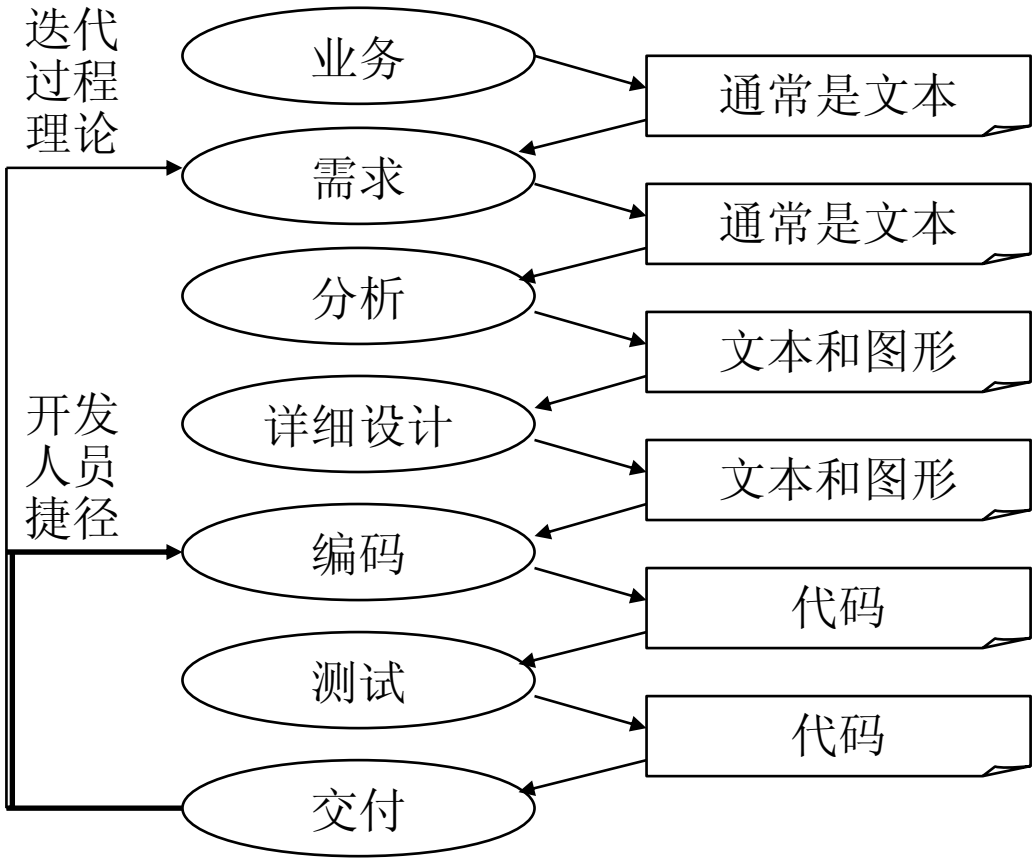
(2) AUTOMATION



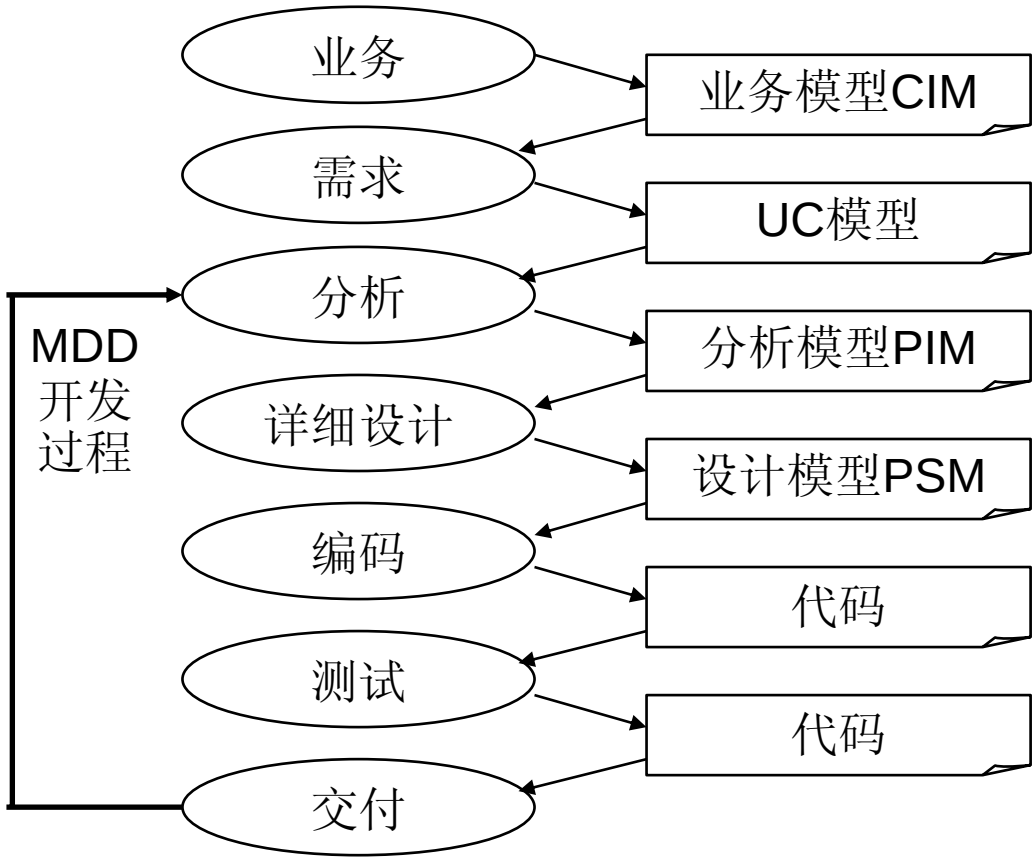
关系到
工具

```
SC_MODULE(producer)
{sc_inslave<int> in1;
int sum; //
void accumulate (){
sum += in1;
cout << "Sum = " <<
sum << endl;}
```

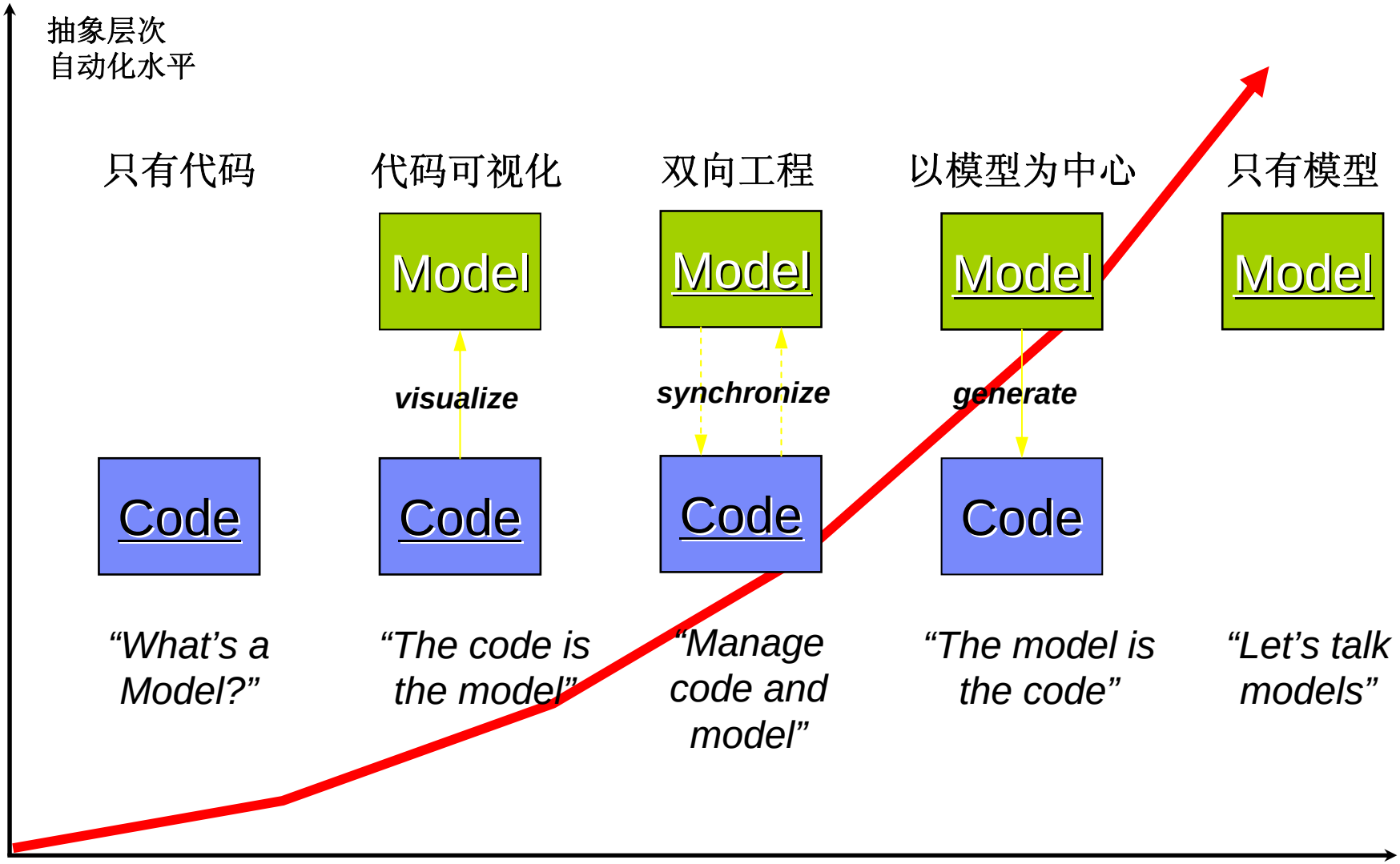
传统的软件开发过程


迭代
过程
理论
开发
人员
捷径
业务
需求
分析
详细设计
编码
测试
交付
通常是文本
通常是文本
文本和图形
文本和图形
代码
代码

MDD的开发过程


业务
需求
分析
详细设计
编码
测试
交付
MDD
开发
过程
业务模型CIM
UC模型
分析模型PIM
设计模型PSM
代码
代码

MDD 的成熟度模型


抽象层次
自动化水平
只有代码
代码可视化
双向工程
以模型为中心
只有模型
Model
Model
Model
Model
visualize
synchronize
generate
Code
Code
Code
Code
"What's a Model?"
"The code is the model"
"Manage code and model"
"The model is the code"
"Let's talk models"

对于模型驱动开发方法的疑问——性能

- 一种”垂直“的模型转换的形式
- 当前的状况:
  - 所有的开发都是通过模型做的 (也就是说，没有对生成的代码进行过修改)
  - Size: 系统相当于 ~ 10 MLoC
  - Scalability: 项目团队有上百的开发人员参加
  - Performance: 与手工编写的系统的差异不超过±5-15%

当前已经达到的程度（MDD）

- 采用完全的自动化的代码生成开发软件的行业：：

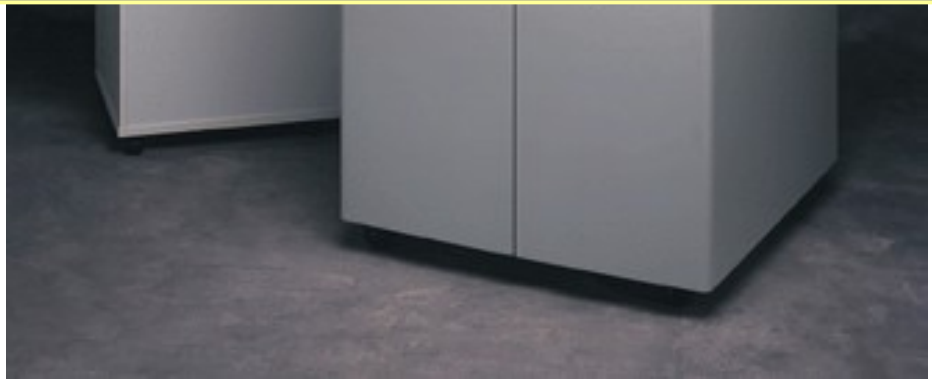
Automated doors, Base Station, Telephone Billing System, Broadband Access, Gateway, Camera, Car Audio, Convertible roof controller, Control Systems, DSL, Elevators, Embedded Control, GPS, Engine Monitoring, Entertainment, Fault Management, Military Data/Voice Communications, Missile Systems, Executable Architecture (Simulation), DNA Sequencing, Industrial Laser Control, Karaoke, Media Gateway, Modeling Of Software Architectures, Medical Devices, Military And Aerospace, Mobile Phone (GSM/3G), Modem, Automated Concrete Mixing Factory, Private Branch Exchange (PBX), Operations And Maintenance, Optical Switching, Industrial Robot, Phone, Radio Network Controller, Routing, Operational Logic, Security and fire monitoring systems, Surgical Robot, Surveillance Systems, Testing And Instrumentation Equipment, Train Control, Train to Signal box Communications., Voice Over IP, Wafer Processing, Wireless Phone

DNA Analyzer (ABI Prism 3700) - PE Biosystems

"这是一个非常复杂的项目，开发的是一个复杂的产品，我们仍然击败了市场上的其它竞争对手."

– Celera Genomics 得到了第一个完整的人类基因序列

- 子系统模型用于与管理层和其他的团队成员沟通设计。每个人都理解了系统都开发了什么，怎样开发的模型的本身就是系统实现。
- 团队成员能够很容易地构造出组件来仿真那些还不存在的硬件和其他的软件系统。在开发阶段就能够验证系统，以确保在其他的组件构造出来之后能够与正在开发的模块集成并且正常工作。
- 一个6个人的团队在18个月的时间内开发出了5 MB的复杂的嵌入式软件代码 (120 lines/person day).
- 按照进度完成了开发任务，并且开发出了高质量的成功的产品



大型电信设备制造商

- 使用了MDD 技术
  - UML, Rational Technical Developer, RUP
- Example 1: 广播基站
  - 200多万行 C++ code (87% 由工具自动生成)
  - 100 多开发人员
- Example 2: 网络控制器
  - 450多万行 C++ code (80% 由工具自动生成)
  - 400多开发人员

带来的好处

减少了80%的缺陷

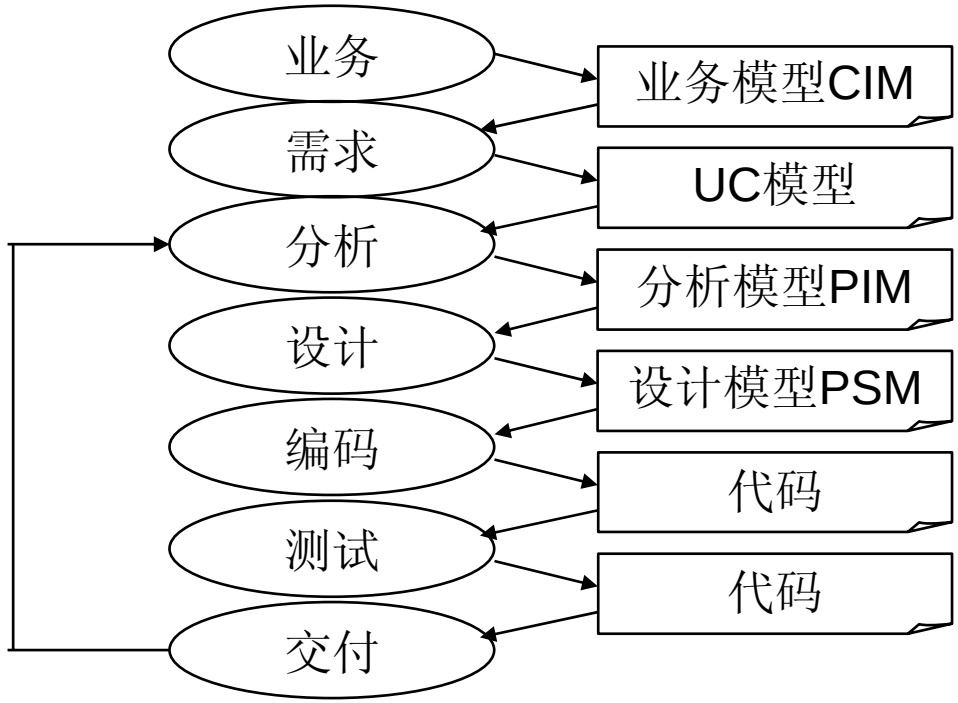
提高了30%的生产效率

更多的实例

- 大型电信设备制造商 1:
  - 代码开发效率从 40 LoC/每天到250 Loc/每天 (>600% 提升)
- 大型电信设备制造商 2:
  - 代码的生产效率从200 LoC/周 到 950 Loc/周 (~500% 提升)
  - 6个人的团队开发出 120 KLoC 的系统在 21.5 周的时间内， 而按照计划是 40 周 (~100% 提升)
  - 缺陷的密度 (每行代码) 降低了 (1700%)

MDD的价值

- 提升了抽象层次
- 实现了架构优先
- 提高团队生产力
- 提高软件产品质量
- 企业资产的持续积累
- **提高企业核心竞争力**



嵌入式系统的传统开发方法——功能分解

- 需求驱动的系统开发方法：
  - 通过功能分解的方式分解系统，建立需求的层次关系
  - 分解系统，将系统分解成为未来可以组装在一起的物理组件
  - 通过功能分配的技术发掘其他的需求并且将它们分配到子系统组件中
  - 逐个层次应用相同的方法，直至需求被分配到足够细化的层次后，开始详细设计
- 这个方法的一个基本的假定是：在开发工作能够顺利进行之前，需求必须被很好地理解。

需要更为行之有效的系统开发方法

- 现代化的系统需要
  - 系统部件之间松散耦合——便于维护与扩展
  - 系统部件之间能够紧密集成——提供高效的运作
  - 进一步减少系统的部件数量
  - 提高每个系统部件的能力

  } 部件之间的集成更为紧密，部件之间的协作方式明确、固定
- 需求驱动的系统开发方法（传统的）不能够适应
  - 以功能分解主导的开发方法使系统倾向于紧耦合，难于维护、难于扩展，应变能力差，易于导致复杂产品开发项目的失败，不便于复用
- MDSD——正是我们期待的系统开发方法
  - 模型驱动的系统开发方法能够让我们开发出更为适应现代要求的系统

”系统“的几个关键概念

- 系统：
  - 一组为提供特定的**服务**而组织在一起的资源。这些服务让系统在与其它系统进行协作的过程中完成被赋予的使命
  - 系统由硬件、软件（包括固件）、工作人员、数据组成
- 系统的特征属性
  - “黑盒”——从外部可以观察到的特征，包括系统能够提供的服务
  - “白盒”——用于组成系统的资源，白盒视图根据所思考的主题呈现系统的内部样态
  - 系统的白盒资源被封装在黑盒的服务中
- 系统开发过程
  - 描述系统的黑盒属性的规格 （例如：系统的需求）
  - 交付满足需求的集成的系统组件 （例如：系统的实现与实施）
  - 实施描述一个部件如何构造或计算
  - 服务是满足需求者的期望的一种机制

MDSD的模型层次

模型层次	表达内容
Context （上下文）	系统的黑盒——系统及其行为人
Analysis （分析）	系统白盒——在每个Viewpoint中系统的最初的分解
Design （设计）	分析层次的模型在资源（硬件、软件和人）中的实现
Implementation （实施）	设计模型在特定的配置下的实现

MDSD的架构框架

	Worker	Logic	Information	Physical	Process	Geometric
Context	角色定义 活动建模	UML 产品上下文模型 UML用例图	UML 企业数据视图，包括扩展产品数据	领域的特定视图		领域的特定视图
Analysis	UML 分配到人、机，活动建模仿真	UML 产品的逻辑分解	产品数据的概念模式	UML 产品的位置视图	UML 产品流程视图	参数化的几何模型 布局
Design	操作指南 帮助文件 工作流、交易管理	软件的组件设计	产品的数据模式	ECM设计	细化的流程视图、时序图	MCAD设计
Implementation	硬件、软件配置					

MDSD的重要原则——分解系统，而不是分解需求

- 系统是一组相互交互、相互关联、相互依赖元素组成的复杂的整体
- 系统可以被抽象成为一个复杂的实体，可以被看作黑盒子
- 出于构造的目的，我们必须打开盒子，看到里面的元素如何协作、如何相互关联、相互依赖的，系统是如何组成——白盒视图
- 系统工程化的过程是从黑盒视图到白盒视图的过程，逐层深入

- 具体的工作步骤
  - 理解系统在其工作环境中的工作模式。将系统当作黑盒子，找出系统的交互对象、实际交互与结果。——上下文层次
  - 决定如何初步将系统分解成为相互交互的元素。递归地从黑盒透视图到白盒透视图地检验每个元素直到全部的关键功能和非功能需求得到满足。——分析层次
  - 考虑系统元素如何被实现 ——设计层次
  - 系统被构建、测试，准备发布 —— 实现层次

系统分解方法

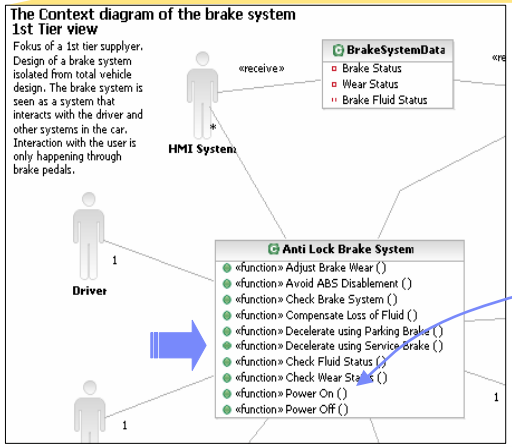
- “系统的系统”
  - 系统封装了提供服务所必要的资源，这些资源也是系统，也各自封装了资源
  - 系统的封装是一个递归的模式，即所谓“系统的系统”
- 系统分解方法
  - 操作分析法——USE CASE分析法的扩展
    - 分解系统，建立系统元素的上下文视图
    - 将系统的操作当作系统元素的用例场景
    - 将系统元素作为黑盒子来描述场景，系统的操作是这些元素的交互实现的
    - 从场景中发现元素的操作
  - 递归地在不同层次应用操作分析方法，实现系统的分解

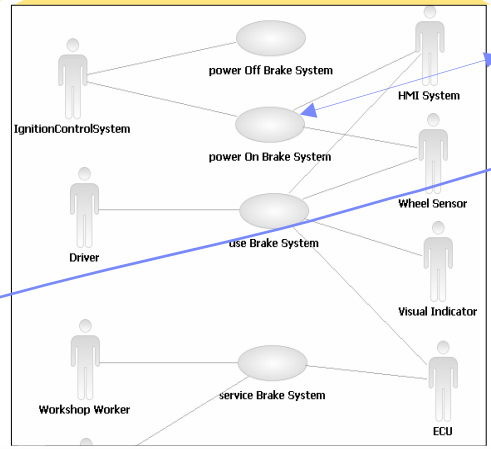
第一步： 明确目标系统的运行环境

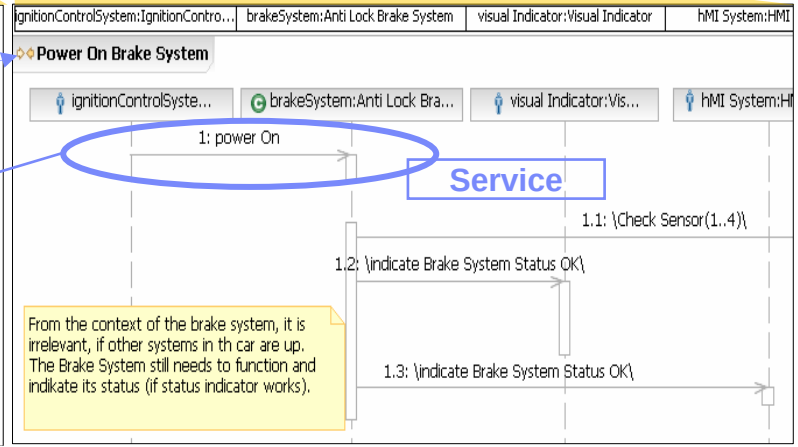
上下文层次视图

上下文层次

上下文概览 → 用例概览 → 用例的详细的流序列图 → 需求分配到用例上







- 哪些部分是系统的部分?
- 系统的边界是什么?
- 存在哪些输入/输出?

- 有哪些人使用系统?
- 他们如何使用系统?

- 在系统所提供的服务被使用时，这些服务看起来是怎样的?
- 系统必须提供哪些服务?

好处

- 改进产品结构，改进与需求的统一，改进需求的满足程度.
- 提高模型以及模型元素的复用程度.

第二步：通过分析分解系统

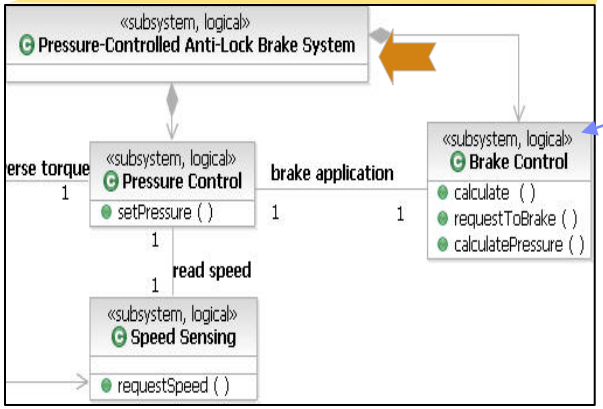
分析层视图

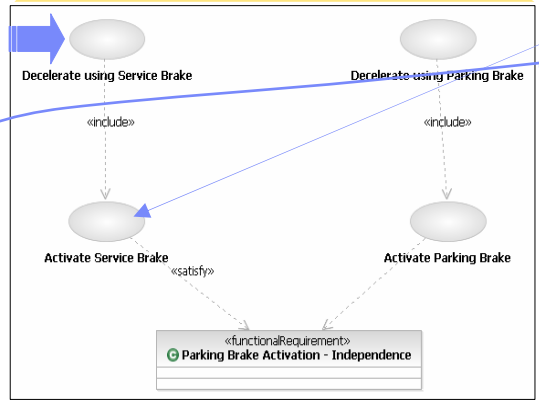
上下文层次的服务映射到分析层次的用例.

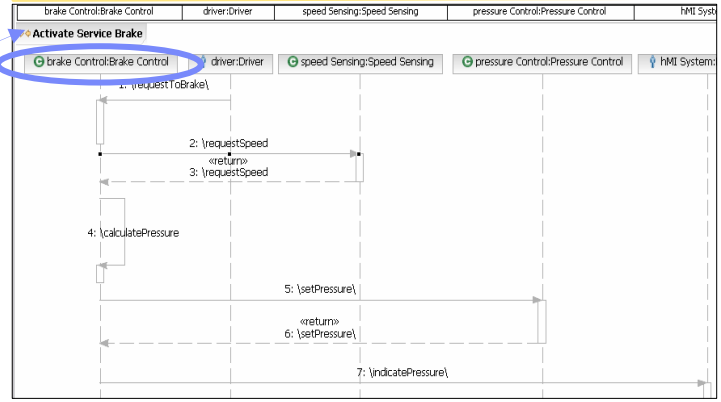
将系统分解到子系统

分析层次

体系结构概览 → 用例概览 → 用例的详细流序列 → 需求分配到用例







- 哪些部分是系统的部分?
- 系统是怎样分解的?

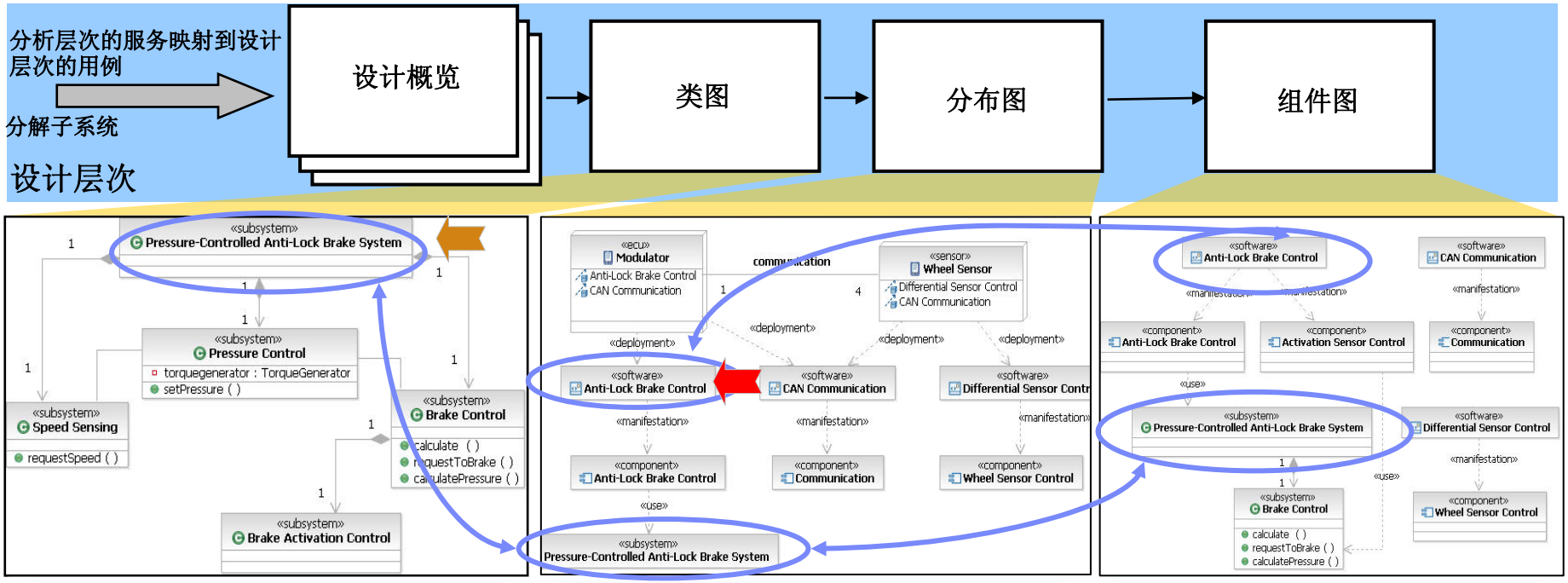
- 谁在使用系统?
- 系统是如何被使用的?

- 在系统被使用时系统的内部是如何工作的?
- 系统必须提供哪些内部的服务?

好处

- 系统得到完整一致的文档化的描述。
- 对于系统的扩展和变更都变得更为容易。

第三步：分解服务、分配到硬件
设计层视图



- 哪些子系统构成软件?
- 子系统都控制什么?
- 子系统需要哪些信息?

- 设备包含哪些组件?
- 设备之间如何连接?
- 在一个节点上部署了哪些组件?

- 子系统在哪里被使用?
- 软件在哪里被使用?
- 系统是如何组成的?

好处

- 不同的配置能够得到一致性的描述并且得到有效的验证。
- 更好地管理系统复杂度。

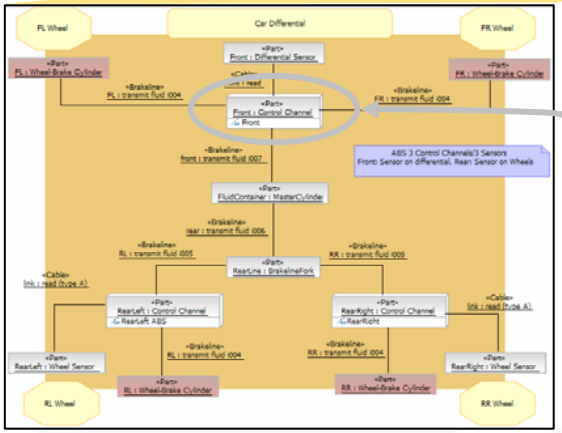
第四步：更为详细的设计
设计层视图

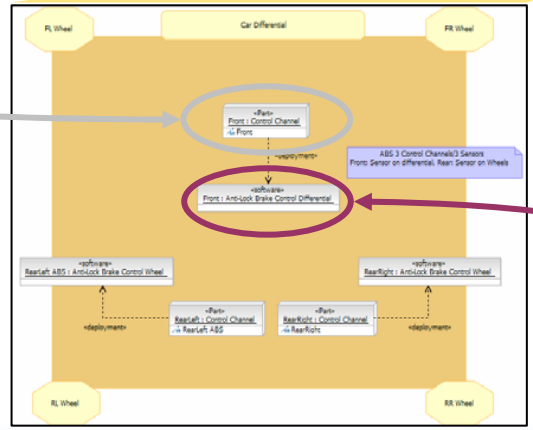
分析层次的服务变成设计层次的用例

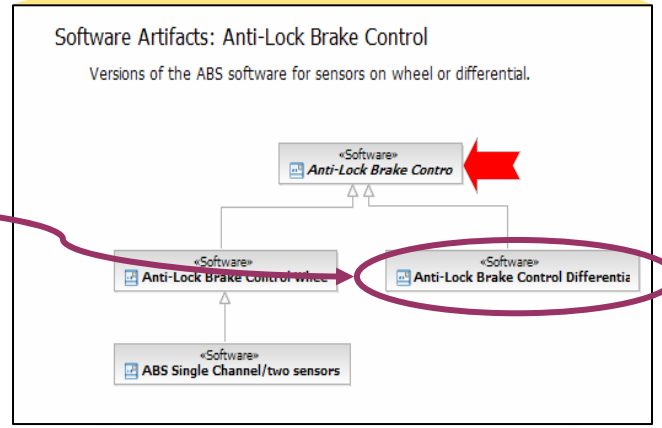
将子系统分解到类

设计层次

体系结构概览 → 组合结构图 → 组合结构图 → 组件图







- 哪些组件定义了系统?
- 哪些组件被用在系统中?
- 系统的不同部分怎样交互?

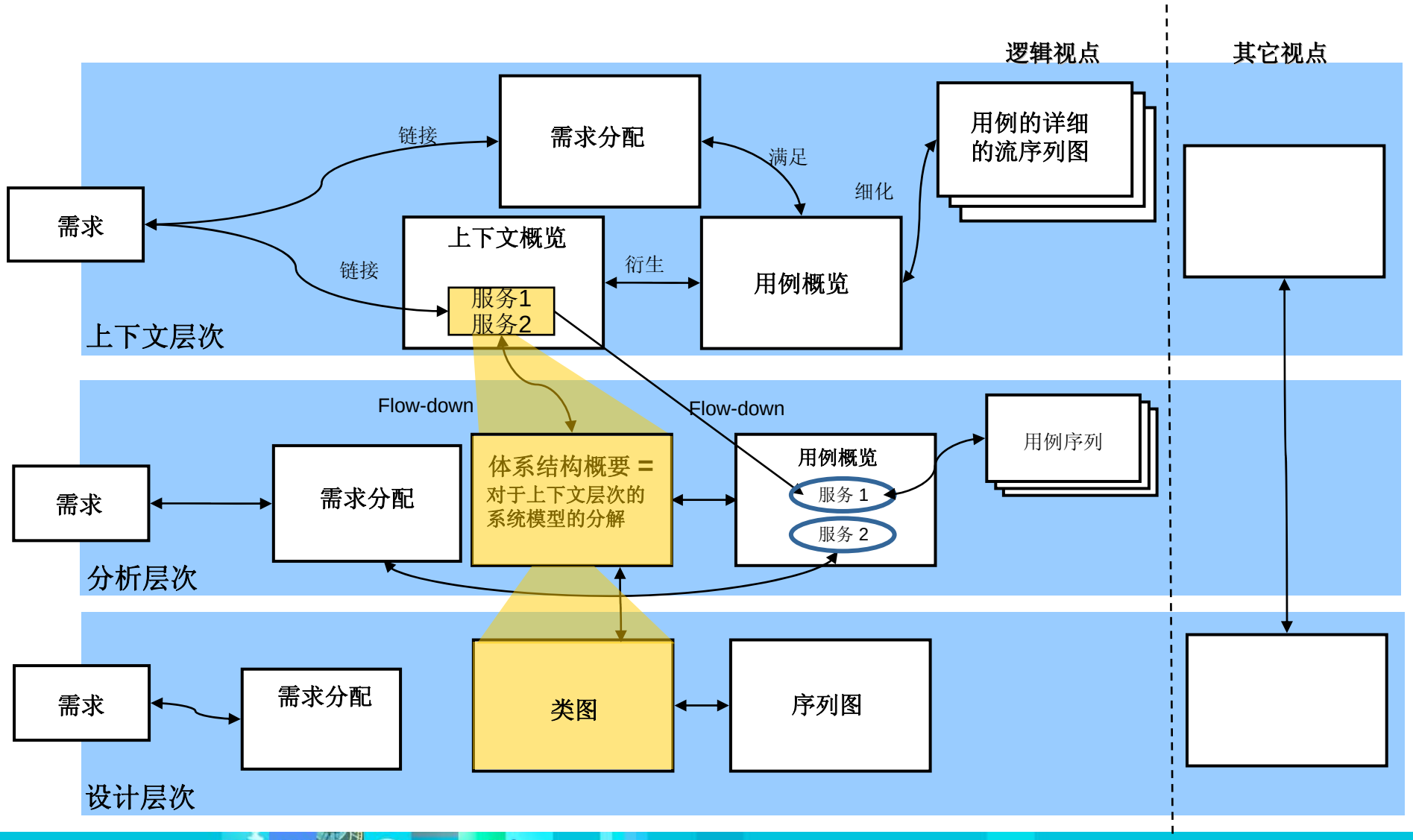
- 系统的一个部分提供哪些服务?
- 一个部分承担哪些角色，它与其他哪些部分之间进行交互?
- 哪些部分是不同部分所需要的公共的部分，并且能够被不同的部分重用?

- 哪些组件的代码，哪些子系统描述了软件的服务?
- 组件之间如何联系的?

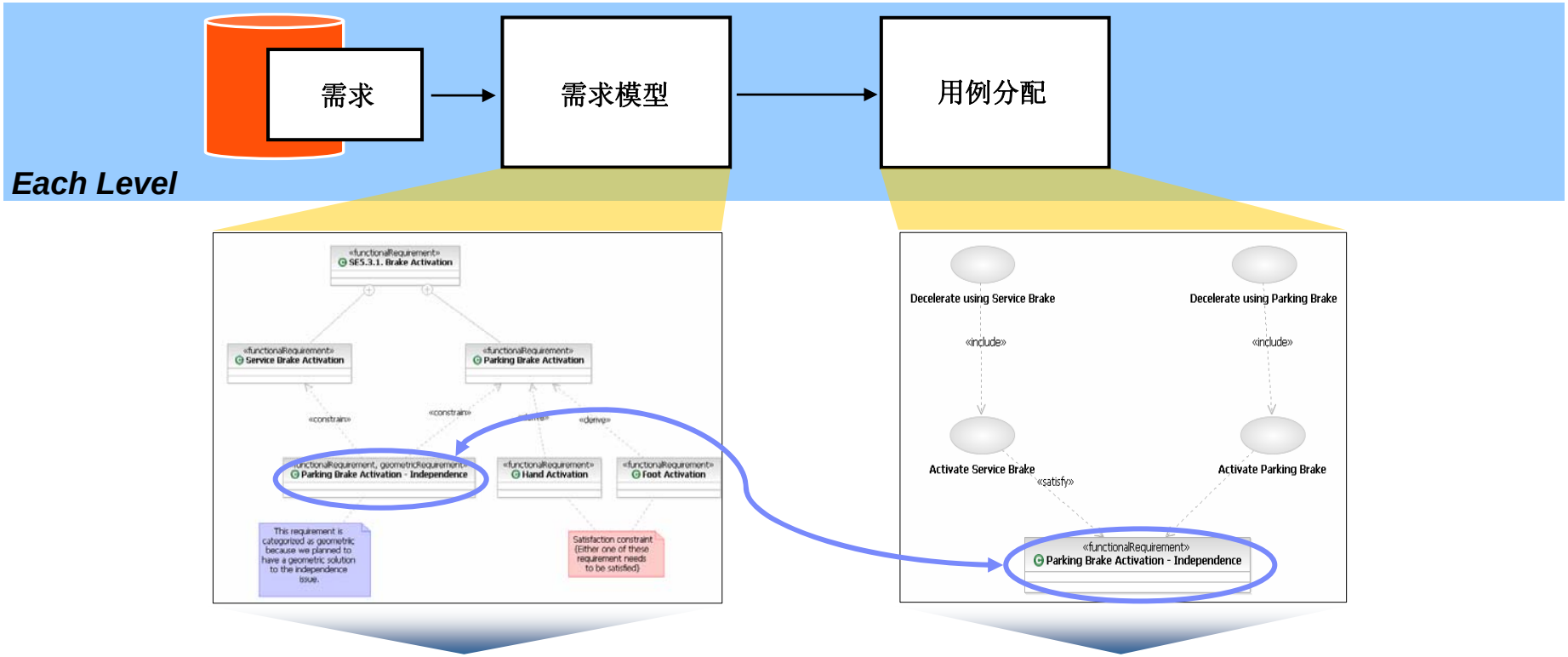
好处

- 不同的配置能够得到一致性的描述并且得到有效的验证。
- 通过利用不同视点的模型易于管理系统的复杂度。

MDSD小结


逻辑视点
其它视点
链接
需求分配
满足
用例的详细
的流序列图
细化
需求
链接
上下文概览
衍生
用例概览
服务1
服务2
上下文层次
Flow-down
Flow-down
用例序列
需求
需求分配
体系结构概要 =
对于上下文层次的
系统模型的分解
用例概览
服务 1
服务 2
分析层次
需求
需求分配
类图
序列图
设计层次

每个层次: 连接需求和模型，并且细化需求和模型



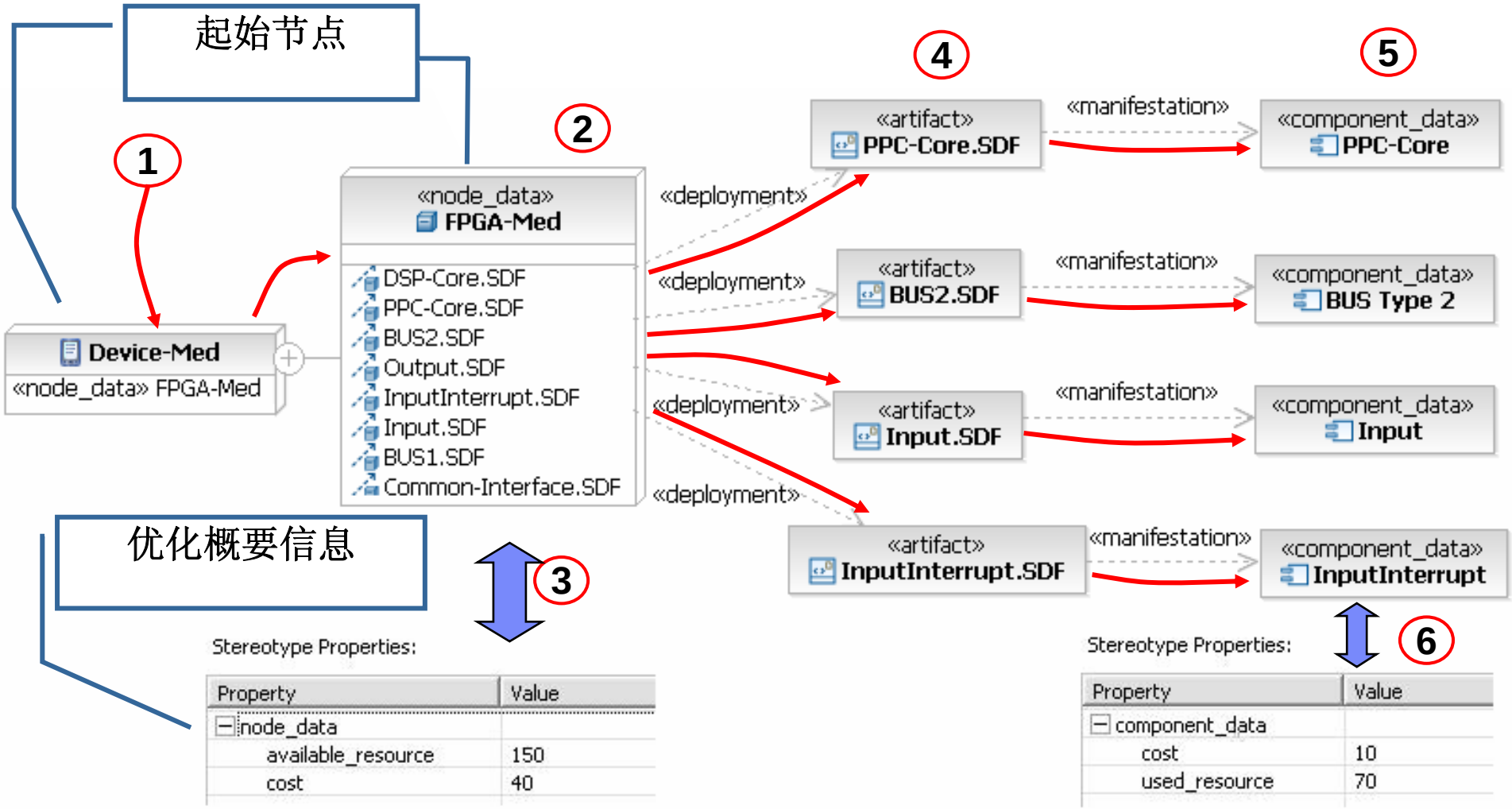
- 哪些需求决定了系统?
- 需求之间是如何关联的?

- 哪些需求被完整地满足了 (例如：用例)?

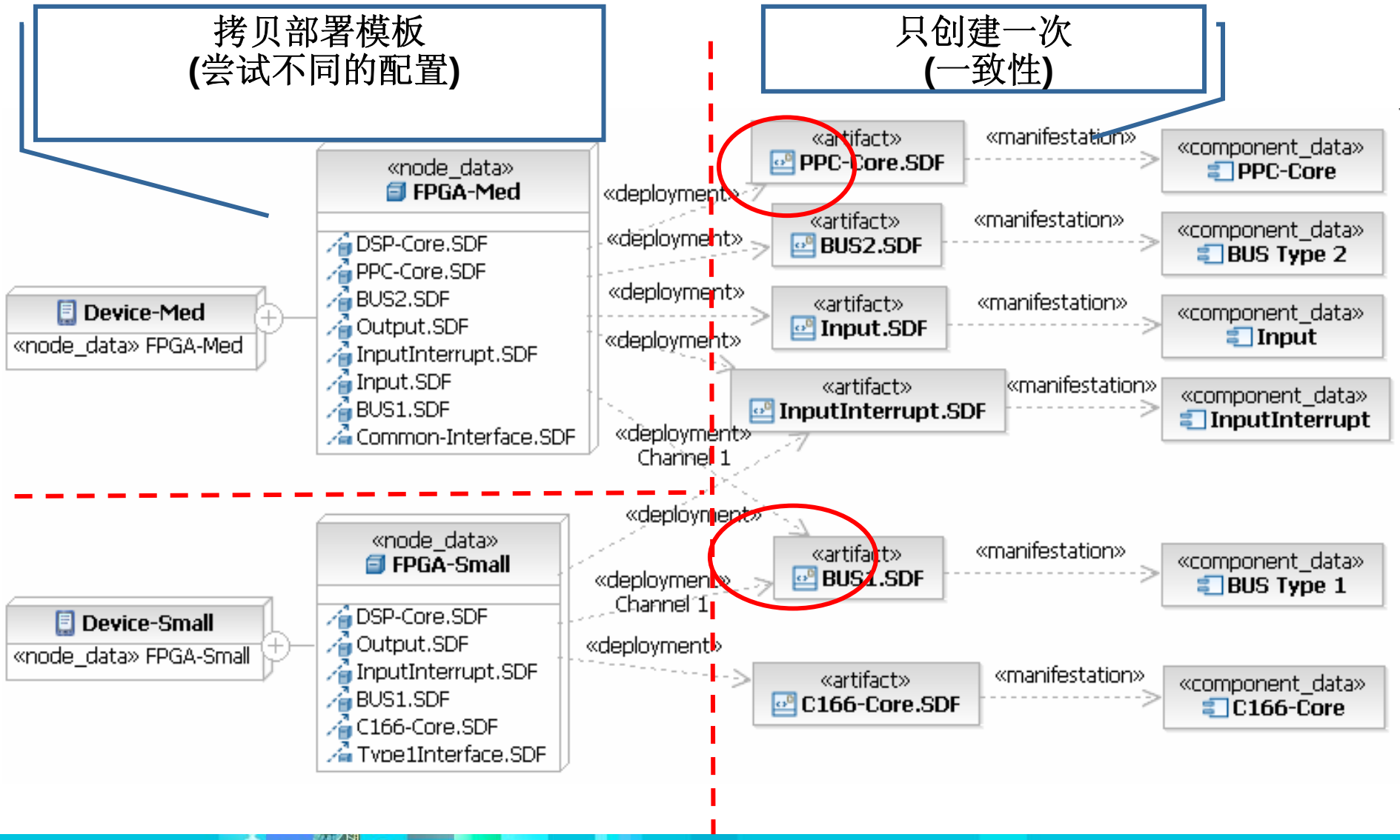
好处

- 需求与模型之间的依赖关系有效地支持了需求的可追踪性。
- 需求与模型层次间的连接关系扩展了需求的可追踪性。

各种部署情况分析 – 原形


起始节点
1
2
«node_data»
FPGA-Med
DSP-Core.SDF
PPC-Core.SDF
BUS2.SDF
Output.SDF
InputInterrupt.SDF
Input.SDF
BUS1.SDF
Common-Interface.SDF
Device-Med
«node_data» FPGA-Med
«deployment»
«deployment»
«deployment»
«deployment»
4
5
«artifact»
PPC-Core.SDF
«manifestation»
«component_data»
PPC-Core
«artifact»
BUS2.SDF
«manifestation»
«component_data»
BUS Type 2
«artifact»
Input.SDF
«manifestation»
«component_data»
Input
«artifact»
InputInterrupt.SDF
«manifestation»
«component_data»
InputInterrupt
优化概要信息
3
6
Stereotype Properties:
Property
Value
node_data
available_resource
150
cost
40
Stereotype Properties:
Property
Value
component_data
cost
10
used_resource
70

概念验证和一致性数据:


拷贝部署模板
(尝试不同的配置)
只创建一次
(一致性)
«node_data»
FPGA-Med
DSP-Core.SDF
PPC-Core.SDF
BUS2.SDF
Output.SDF
InputInterrupt.SDF
Input.SDF
BUS1.SDF
Common-Interface.SDF
Device-Med
«node_data» FPGA-Med
«node_data»
FPGA-Small
DSP-Core.SDF
Output.SDF
InputInterrupt.SDF
BUS1.SDF
C166-Core.SDF
Type1Interface.SDF
Device-Small
«node_data» FPGA-Small
«deployment»
Channel 1
«artifact»
PPC-Core.SDF
«artifact»
BUS2.SDF
«artifact»
Input.SDF
«artifact»
InputInterrupt.SDF
«artifact»
BUS1.SDF
«artifact»
C166-Core.SDF
«manifestation»
«component_data»
PPC-Core
«component_data»
BUS Type 2
«component_data»
Input
«component_data»
InputInterrupt
«component_data»
BUS Type 1
«component_data»
C166-Core

各种部署情况分析 – 结果

Tasks | Console | Snippets

Pluglets

```
>------------Total: Node----------------
Device-Med
>--------Calculationg Node--------------
  Device-Med
     Nodes start...
      >Device-Med
      >FPGA-Med
     Nodes end...
     Components start...
     > DSP-Core
     > PPC-Core
     > BUS Type 2
     > Output
     > InputInterrupt
     > Input
     > BUS Type 1
     > Common-Interface
     Components end...


----------------------------
------ Node Analysis --------
    Device-Med
----------------------------
 Total Cost ..........: 40.0
 Total Resources Available : 150

----------------------------
---- Deployed Components ----
----------------------------
 Total Cost ........: 30.0
 Total Used Resources : 168

 Total Cost Nodes+Deployments : 70.0
```

Tasks | Console | Snippets

Pluglets

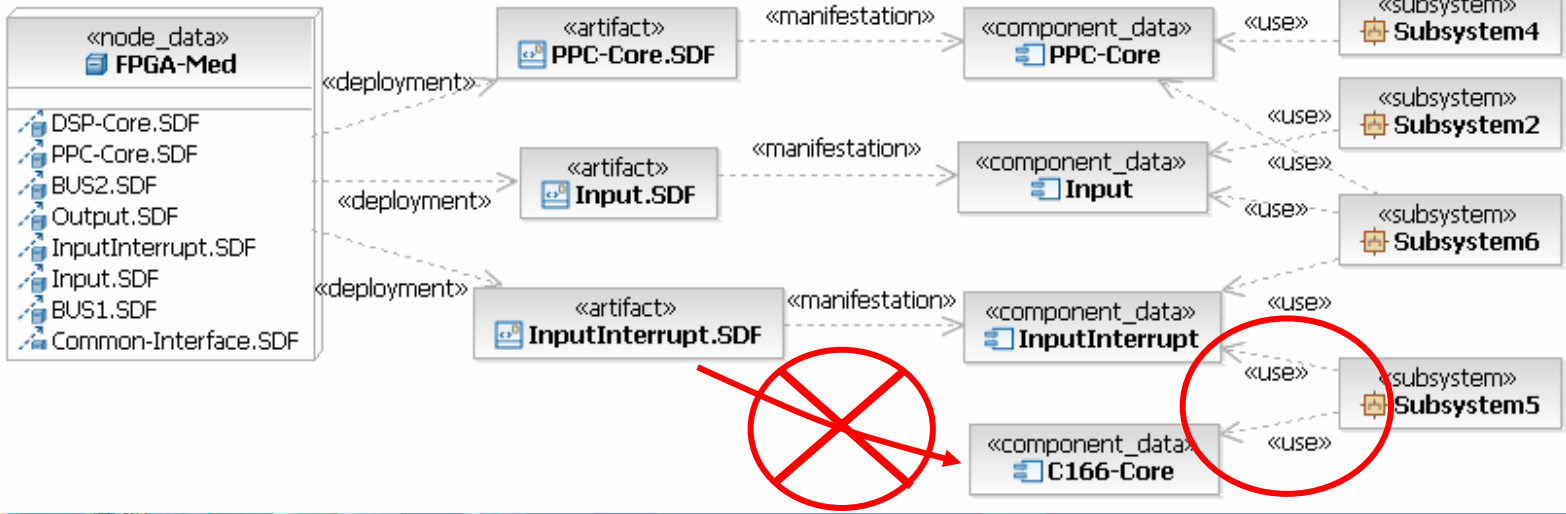
```
----------------------------
------ Free Ressources ------
----------------------------
!!!!!!!!! Available Ressources Exceeded !!!!!!!!!!!!
 Free resources .....: -18
```

- 计算不同节点和组件的开销 (例如：许可证开销)
  - 每个节点/组件的开销
  - 全部的开销
- 计算资源的可靠性
  - 内存
  - CPU循环
  - 芯片管脚
  - 芯片尺寸
  - 其它....

通过“使用” 的依赖关系形成的存在分析

```
Pluglets
-------------------------
---- Realizable Systems ----
-------------------------
   ->Checking SubSystem: Subsystem2
   ->Checking SubSystem: Subsystem5
   ->SubSystem missing components: Subsystem5
       missing component: C166-Core
   ->Checking SubSystem: Subsystem3
   ->Checking SubSystem: Subsystem7
   ->SubSystem missing components: Subsystem7
       missing component: Type1-Interface
   ->Checking SubSystem: Subsystem1
   ->SubSystem missing components: Subsystem1
       missing component: Type1-Interface
   ->Checking SubSystem: Subsystem4
   ->Checking SubSystem: Subsystem6
 Subsystem2
 Subsystem3
 Subsystem4
 Subsystem6
```

- 子系统依赖于组件可用性
- 通过组件的使用实现的子系统
- 通过检查已经部署的组建来验证那个子系统可以被实现



The IBM Rational Software Development Platform

Analyst

对于业务需求建模
识别业务中的服务

Architect

设计服务的体系架构

分析可以被复用的资产

Developer

创建服务

集成并交付组合后的应用

Tester

测试每个独立的服务
测试组合后的应用

Deployment Manager

供应、配置、调整、维护组合后的应用

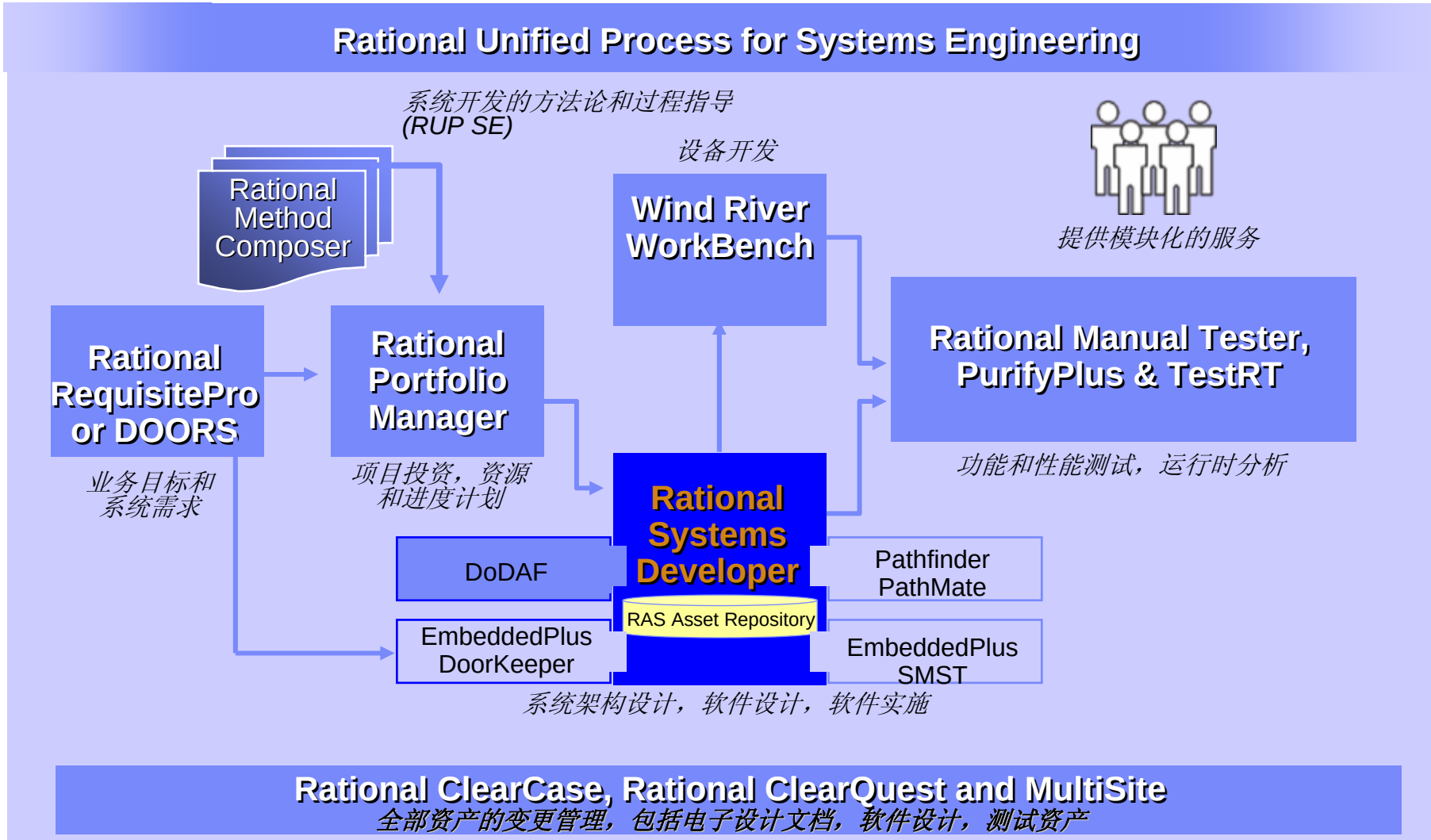
Project Manager

- 管理业务流程
- 管理需求
- 管理资产
- 管理质量

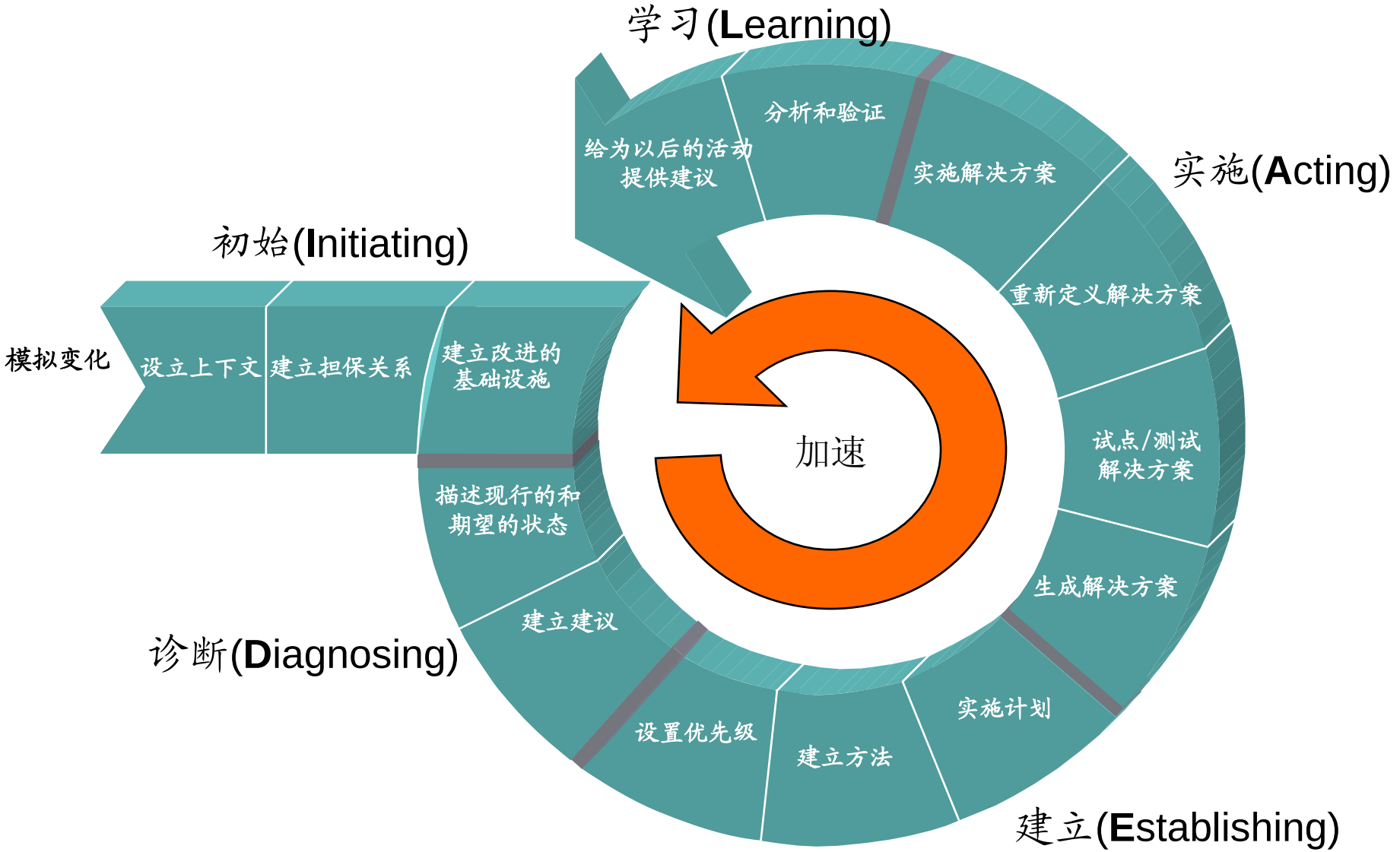
IT Executive

- 确保项目投入符合业务发展需要
- 掌控投资、资源使用

RSD 作为框架 (工具平台)

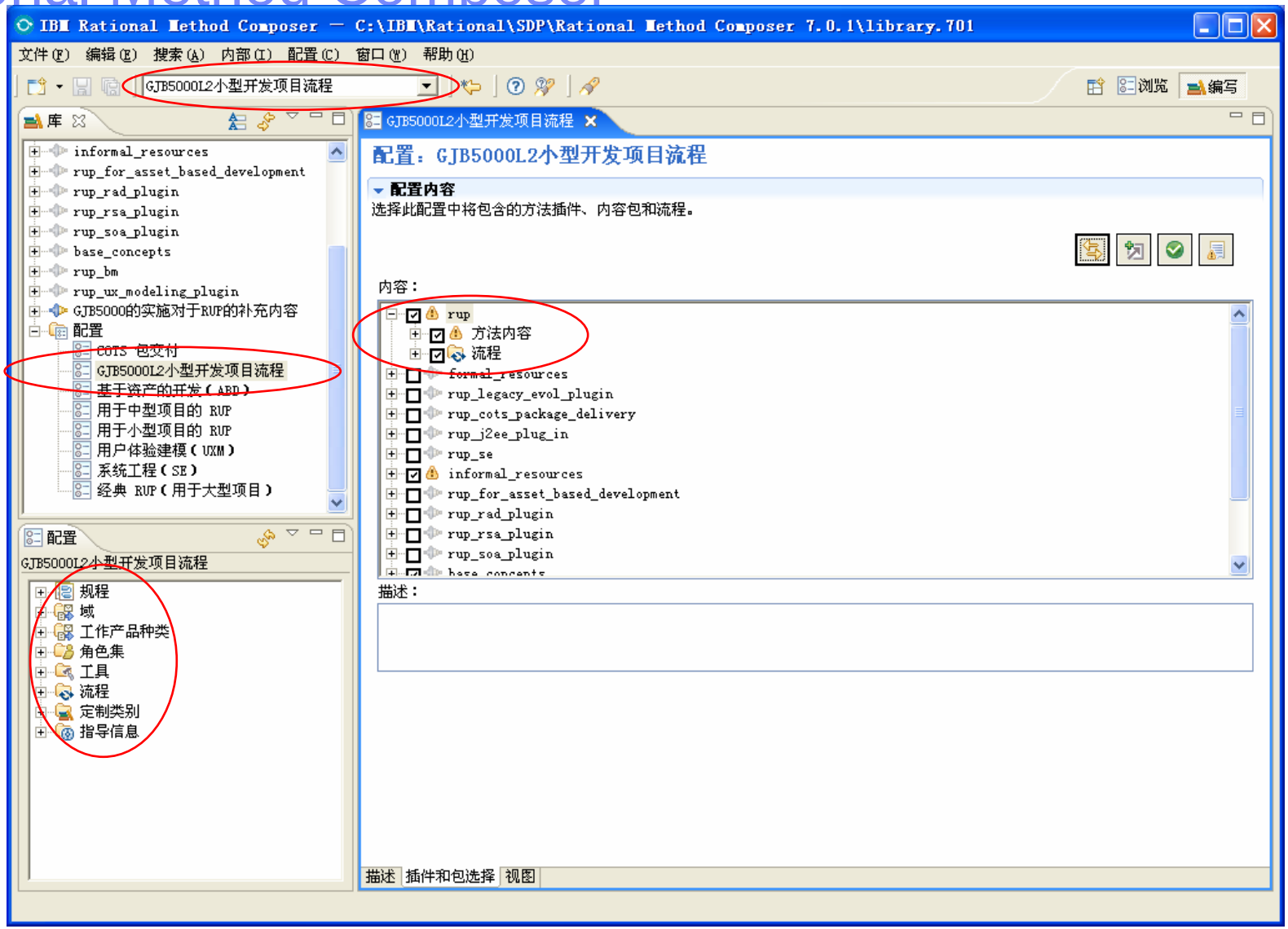


GJB5000实施——软件过程改进IDEAL模型



Rational Unified Process

Rational Method Composer



IBM Rational Portfolio Manager

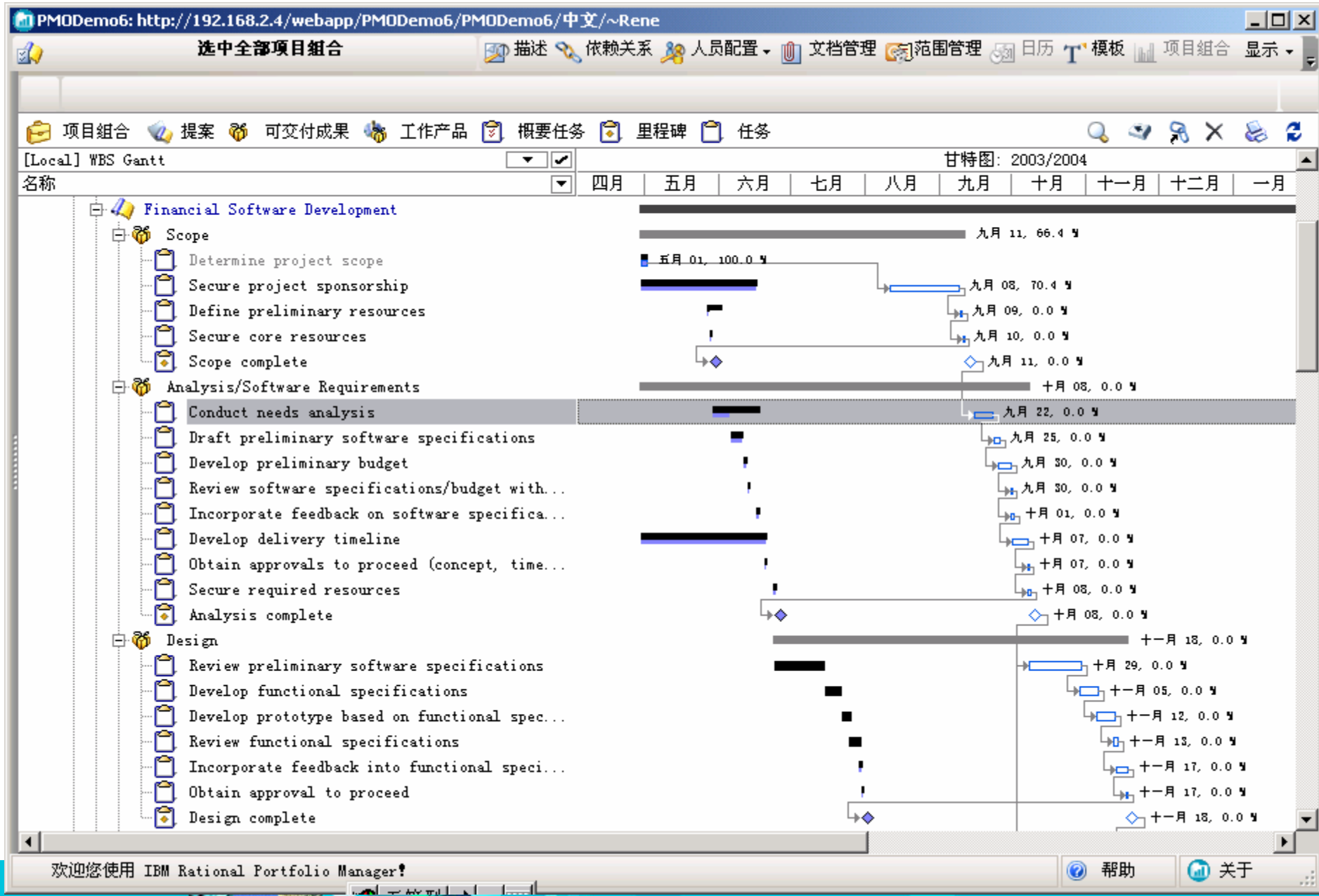
把所有的工作整合在一个流程和平台上

- Scope Management
- Proposal Management
- Portfolio Management
- Resource Management
- Project Management
- Work Management
- Time & Expense Tracking
- Exception Management
- Service Request Management
- Issues Management

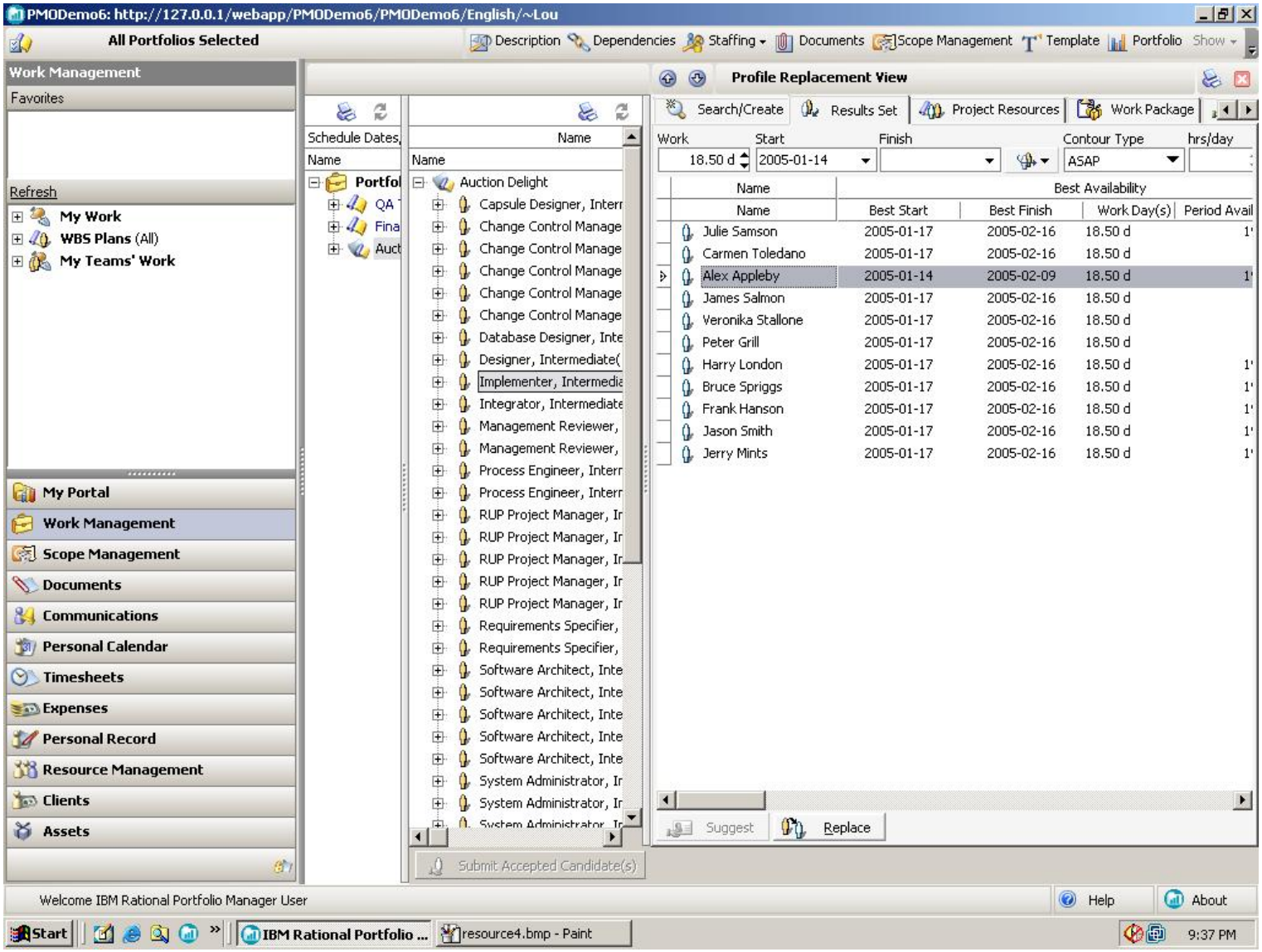
监控企业内部所有项目的状况

监控企业内部所有项目的状况

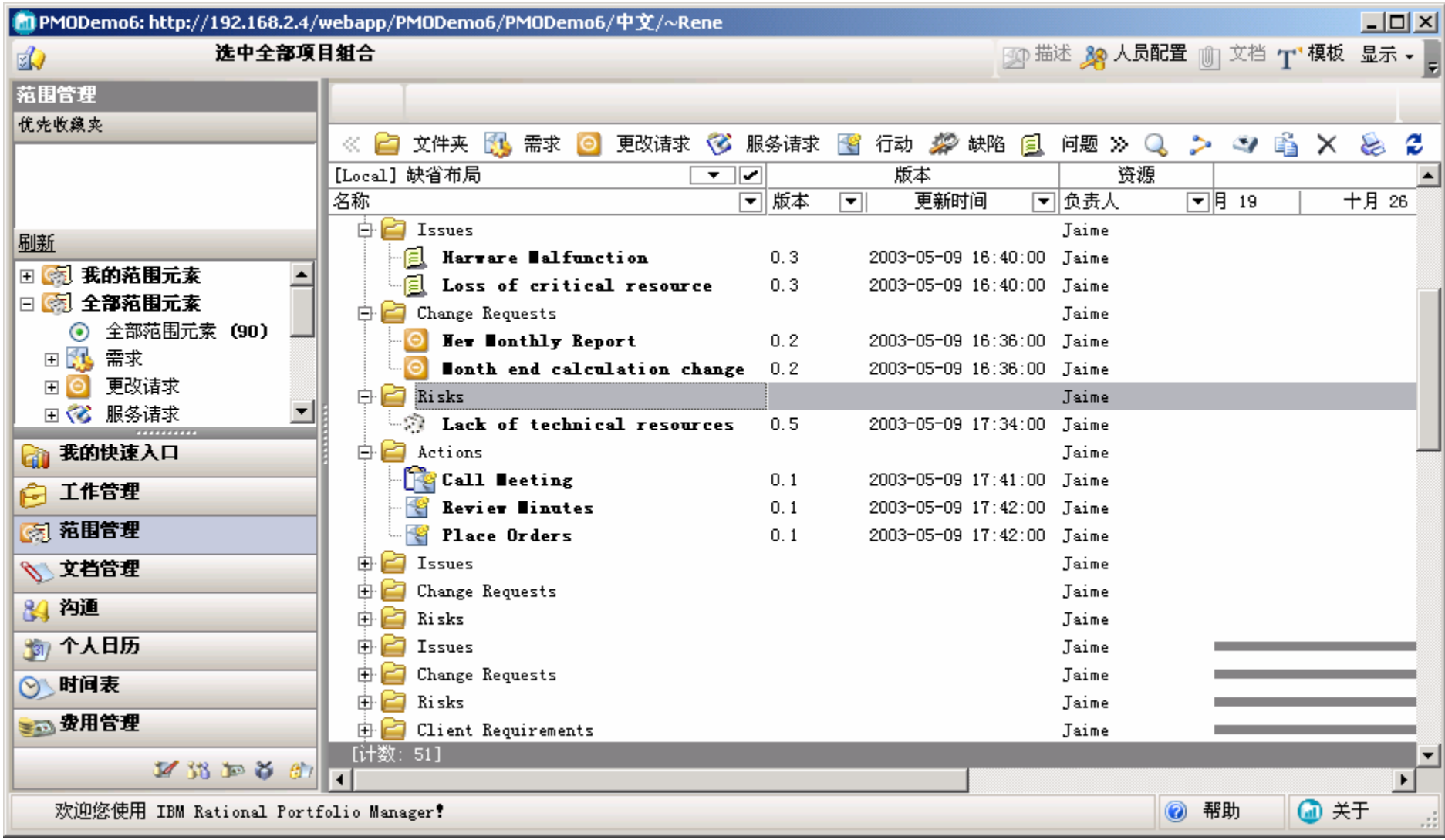
项目经理——管理WBS结构



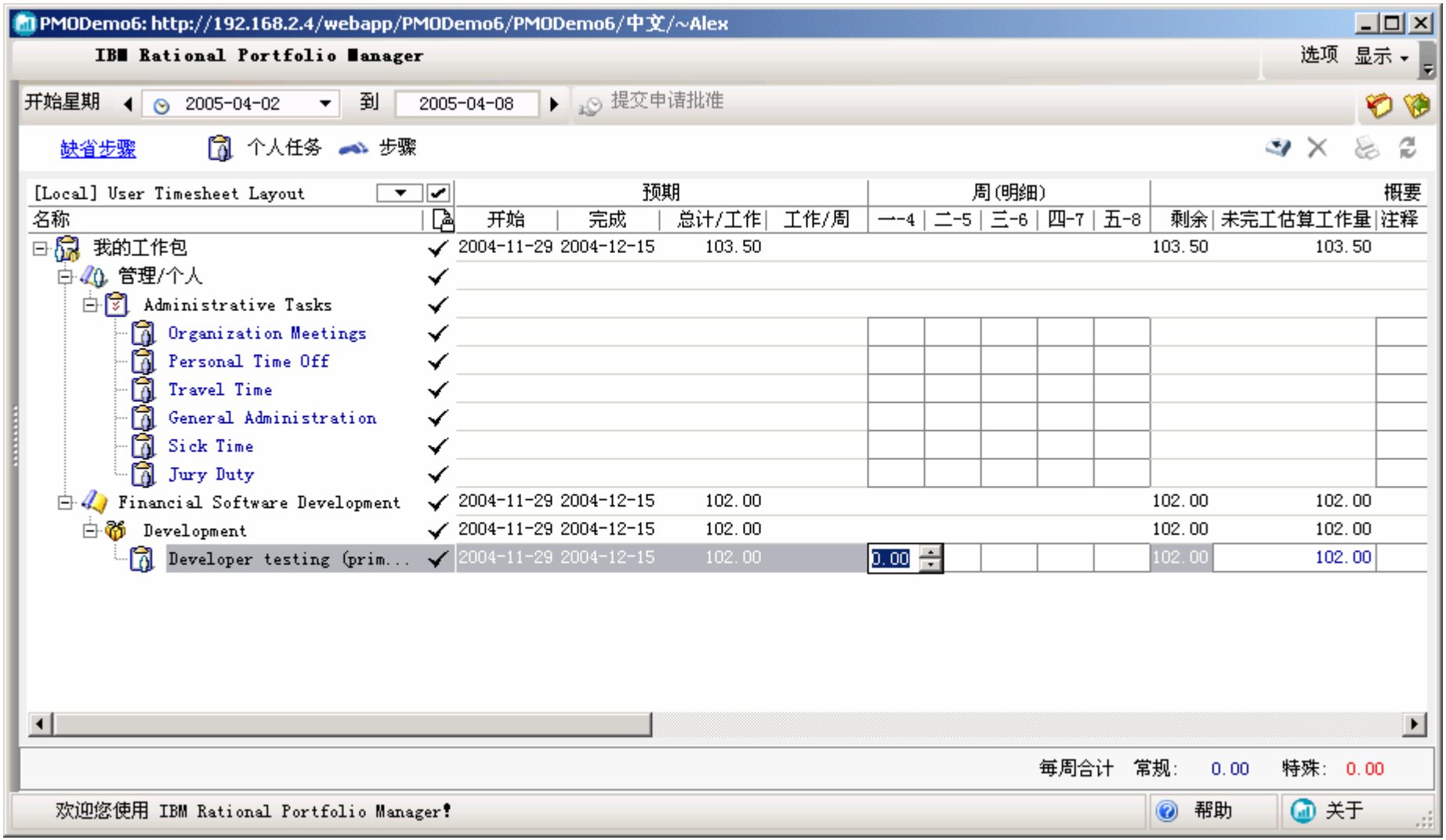
项目经理 – 分配项目资源


PMODemo6: http://127.0.0.1/webapp/PMODemo6/PMODemo6/English/~Lou
All Portfolios Selected
Description
Dependencies
Staffing
Documents
Scope Management
Template
Portfolio
Show
Work Management
Favorites
Refresh
My Work
WBS Plans (All)
My Teams' Work
My Portal
Work Management
Scope Management
Documents
Communications
Personal Calendar
Timesheets
Expenses
Personal Record
Resource Management
Clients
Assets
Profile Replacement View
Search/Create
Results Set
Project Resources
Work Package
Work
Start
Finish
Contour Type
hrs/day
18.50 d
2005-01-14
ASAP
Schedule Dates,
Name
Portfol
QA
Fina
Auct
Auction Delight
Capsule Designer, Interr
Change Control Manage
Database Designer, Inte
Designer, Intermediate(
Implementer, Intermedia
Integrator, Intermediate
Management Reviewer,
Process Engineer, Interr
RUP Project Manager, Ir
Requirements Specifier,
Software Architect, Inte
System Administrator, Ir
Best Availability
Best Start
Best Finish
Work Day(s)
Period Avail
Julie Samson 2005-01-17 2005-02-16 18.50 d
Carmen Toledano 2005-01-17 2005-02-16 18.50 d
Alex Appleby 2005-01-14 2005-02-09 18.50 d
James Salmon 2005-01-17 2005-02-16 18.50 d
Veronika Stallone 2005-01-17 2005-02-16 18.50 d
Peter Grill 2005-01-17 2005-02-16 18.50 d
Harry London 2005-01-17 2005-02-16 18.50 d
Bruce Spriggs 2005-01-17 2005-02-16 18.50 d
Frank Hanson 2005-01-17 2005-02-16 18.50 d
Jason Smith 2005-01-17 2005-02-16 18.50 d
Jerry Mints 2005-01-17 2005-02-16 18.50 d
Suggest
Replace
Submit Accepted Candidate(s)
Welcome IBM Rational Portfolio Manager User
Help
About
Start
IBM Rational Portfolio ...
resource4.bmp - Paint
9:37 PM

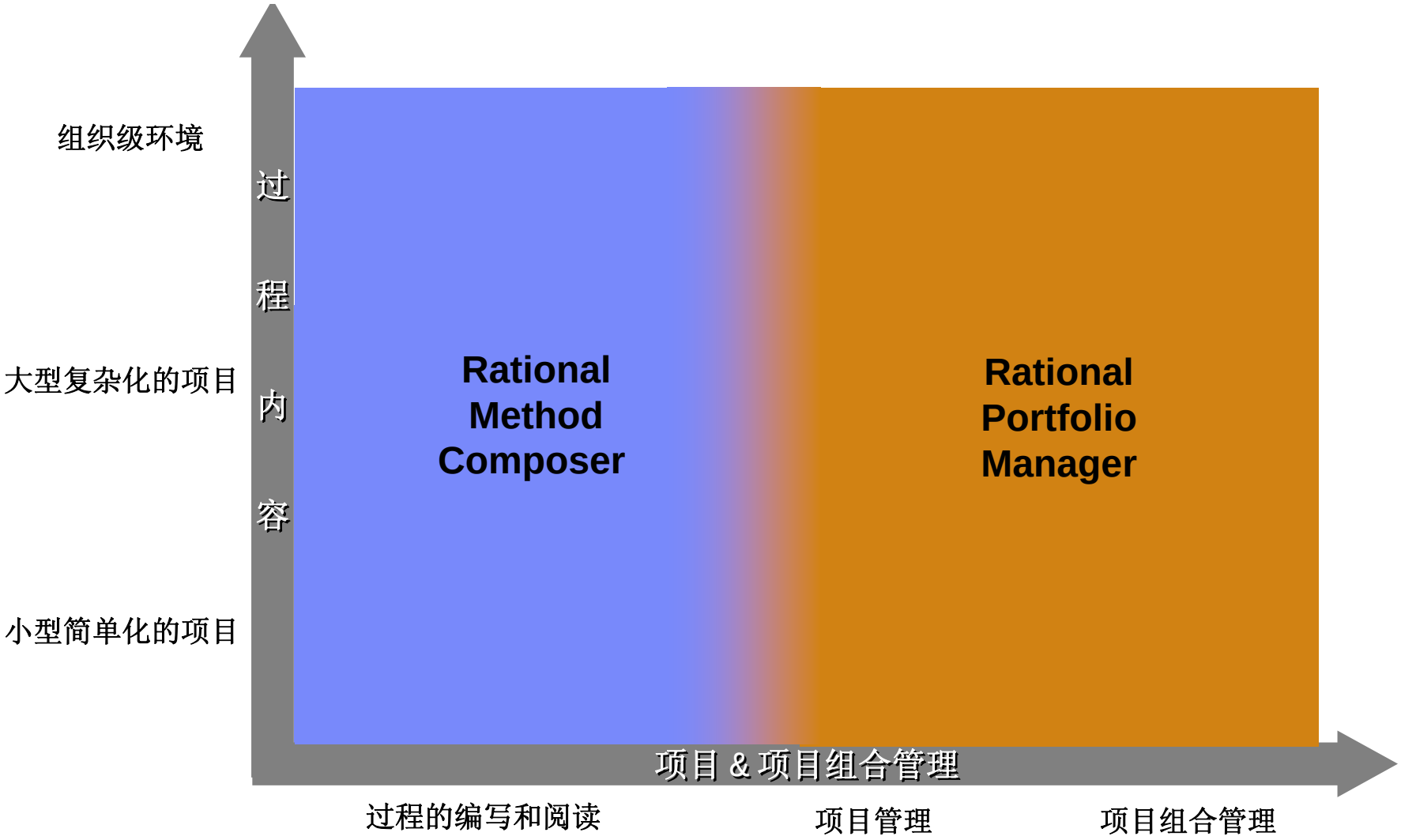
项目经理 – 管理项目中的风险和异外

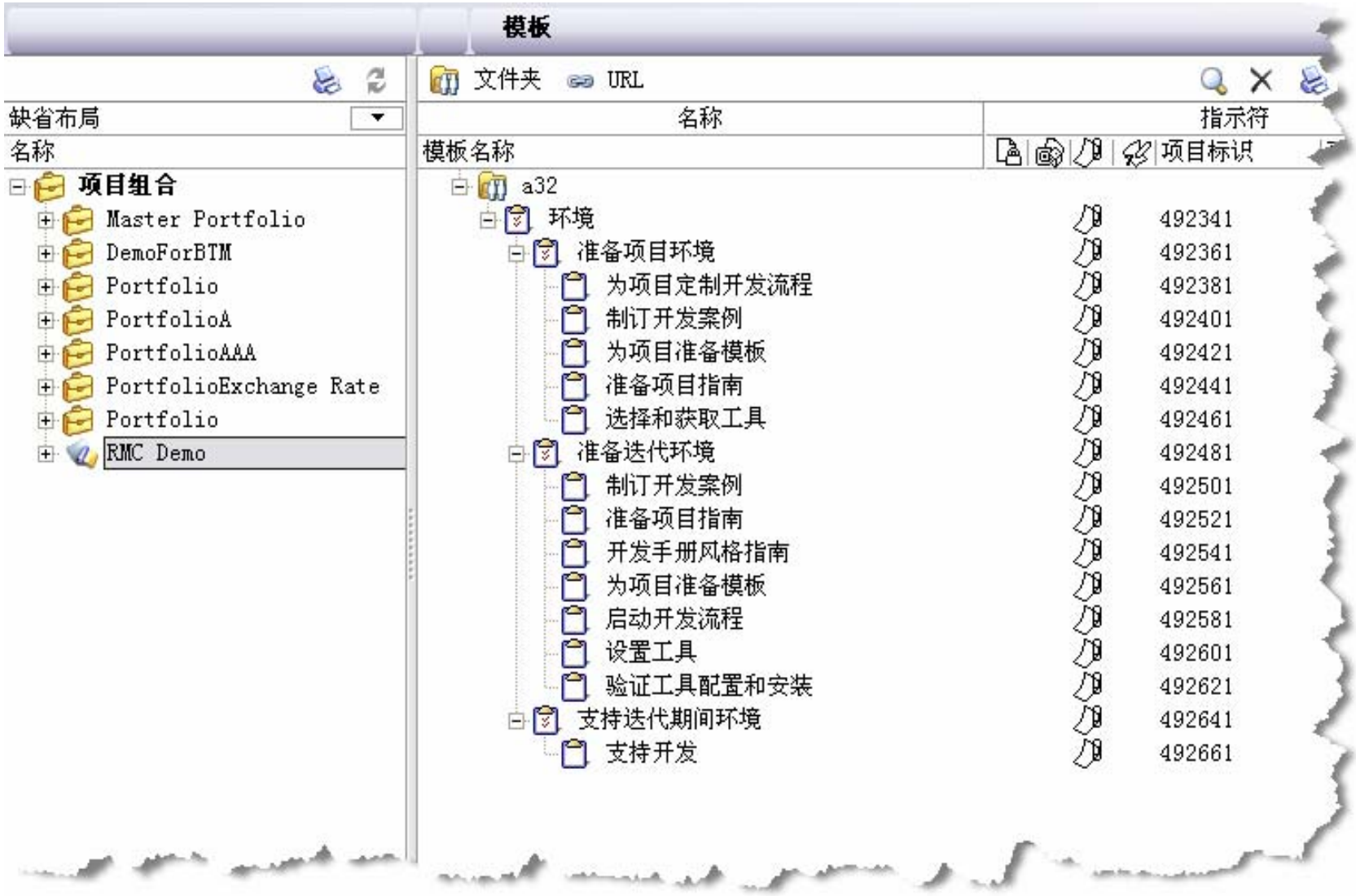


项目组成员 – 汇报个人每周的工作情况



今天


组织级环境
过程内容
大型复杂化的项目
小型简单化的项目
Rational Method Composer
Rational Portfolio Manager
项目 & 项目组合管理
过程的编写和阅读
项目管理
项目组合管理


模板
文件夹
URL
缺省布局
名称
项目组合
Master Portfolio
DemoForBTM
Portfolio
PortfolioA
PortfolioAAA
PortfolioExchange Rate
Portfolio
RMC Demo
名称
指示符
模板名称
项目标识
a32
环境 492341
准备项目环境 492361
为项目定制开发流程 492381
制订开发案例 492401
为项目准备模板 492421
准备项目指南 492441
选择和获取工具 492461
准备迭代环境 492481
制订开发案例 492501
准备项目指南 492521
开发手册风格指南 492541
为项目准备模板 492561
启动开发流程 492581
设置工具 492601
验证工具配置和安装 492621
支持迭代期间环境 492641
支持开发 492661

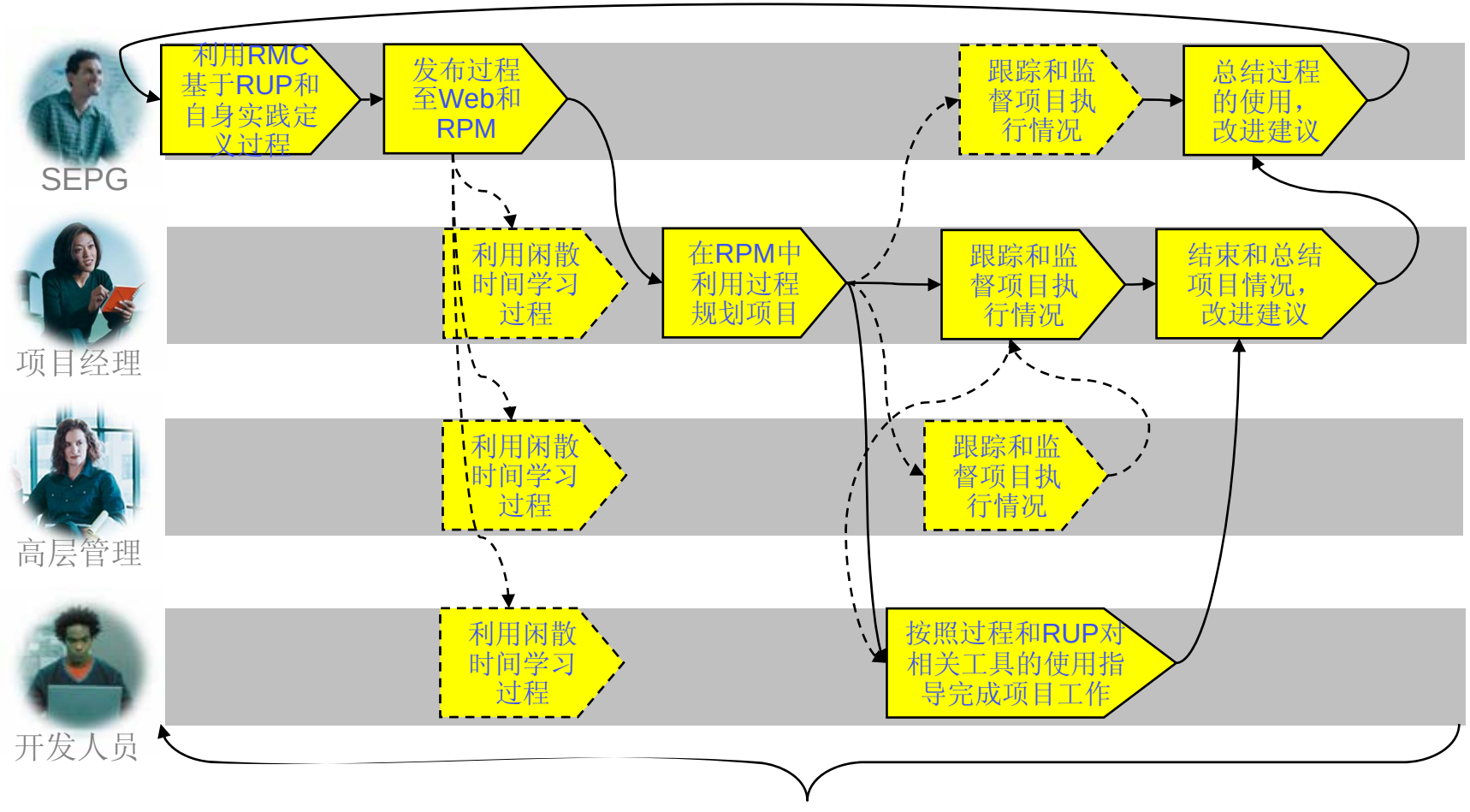
RUP、RMC和 RPM

Proven. Practical. Flexible.

- RUP一个成熟、完整、先进、经过验证的软件开发最佳实践库，蕴含着很对能够直接采用的最佳实践，能够指导和帮助软件开发组织有效改进自身最佳实践，使之与业界先进水平靠拢，在此基础上自主创新。（Rational与IBM方法体系的融合）
- Rational Method Composer 是一个可配置的过程平台，它能够提供RUP的内容作为过程内容基础，同时提供方便的工具支持自主的过程定制，通过简单的配置，就能够定制出任意的过程发布。还可以向内容库中补充自定义内容，对RUP进行补充和扩展。
- Rational Portfolio Manager是一个集成一体化的项目和项目组合管理工具，提供完整的项目综合管理能力。能够支持项目管理人员基于项目过程模版规划项目，方便过程在项目中的使用；同时提供强大的项目跟踪与监督能力，同时提供项目的质量管理、需求管理、风险管理等多方面的能力。

RUP、RMC和 RPM

在业界最佳实践的基础上提供GJB5000的闭环自改进体系




Questions


Thank
You