X-Programmer 总第4期
软件工程的播种机
www.umlchina.com
非程序员 2001 8

目录

方法

过程

本期审稿：Think，June，Windy

联系方法

投稿：editor@umlchina.com
广告：adv@umlchina.com
反馈：think@umlchina.com

播种机：http://www.umlchina.com

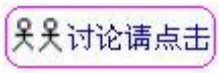
征稿

关于需求，设计，构造，测试，维护，配置管理，管理，过程，工具，质量... 原创或翻译均可。

请点击查看详情>>

征稿：
Using Patterns to Integrate UML Views
Interface Hall of Shame
GOF patterns for GUI Design

请点击查看详情>>



用例的使用误区：管理需求之一

Dr. Timothy Korson 著，James Zhong 译

如果我发现另一个小组以“用例驱动方法”名义如此行事，真想在下届 Object Expo（对象博览会）时，由一支行刑队带着蒙着眼睛的整个小组去面对公众执行枪决！

根据我的经验，我发现“用例”（Use Case）被误用的情形比正确使用更常见。更糟糕的是，这样常常导致严重后果，并与良好的面向对象开发的要点相冲突。由此误用导致的一些常见问题如下：

- 低质量的的需求
- 低质量的设计，不外乎是披着“对象外衣”的“功能分解”
- 时间与精力的浪费

以下案例展示了上面最后一点。

这是个很大规模的项目。可能是所经历的最大的面向对象项目之一。一方面，我听一位高级经理抱怨说有遍及三大洲、超过 1000 名的软件开发人员，正为此项目而工作。项目卷入了基于一个多层次框架下的大量应用程序的准同步开发，而由好胜、顽固者组成的小组是对象技术的新手！

鉴于项目的规模及复杂性，几乎没有错误的开头或没有挫折是令人称奇的。然而，一件轶事深印于我的脑海。项目中的一个福尔摩斯迷这样提及它：无用用例的案例。在项目早期，公司总部搭建了一支小组开始框架开发。最初的小组活动集中于

- 构建领域模型，以及
- 根据相似的传统系统所得的经验，争论必须要在新的面向对象框架中解决的领域特有的构架问题。

随着框架工作的进展，构架小组意识到，他们需要理解将基于这个框架而创建的应用的需求范畴。因此，既然“用例”是现在的时髦用语，一个电话打到相应的应用开发小组，告诉他们：“把你们的

用例发送过来”。作为响应，分布于世界各地的公司机构的应用开发小组都买了 Ivar Jacobson 的书：《面向对象的软件工程》（Object-Oriented Software Engineering），《用例驱动的方法》（A Use Case Driven Approach）；研读贩卖机示例，并写下成百上千个用例！最终，公司总部的框架小组则得到类似于 12,386 个用例这样的产物。

不幸的是，这些用例对框架小组不是非常有用——不是因为他们不需要其中的信息——而是因为信息处于错误的抽象级别上。快速的草稿式估算发现产生出那 12,386 个无用的“用例”底线费用已超过了 50 万美元。令人吃惊吧？并且，应用开发小组的士气付出了高昂代价，框架小组的自信亦被侵蚀。小组成员投入了大量的时间和精力学习并开发“用例”，却仅仅得到自己的工作被忽视的结局。在这种情形下真正可悲的部分是，原始的“用例”通常不仅太详细，还不时充满了错误。当项目后期需要这些详细用例时，它们几乎总是要重做。这只是我目击的众多情形中的一个，出现的原因是对需求过程既没有很好理解也没有恰当的管理。

对如下图表考虑片刻。位于左边的栏处理业务需求及界面规格说明书。位于右边的栏处理如软件设计及源代码等可交付品。极少的开发小组会从直接跳到编写代码处开始一个多年的项目。即使最未经受训练的小组也会做某些级别的领域分析，并在开始编码前设计。

既未很好被理解又未被接受的是，很多抽象级别在左边栏也需要。从详细规格说明书级别开始来获取正确的需求，比从源代码级别开始编写设计得当的正确的系统， 更不可能！如何收集需求而不会对项目导致严重后果，存在着不可忽视的基本原则。

需求工件	开发工件
业务需求 最初很低级别的用例	领域模型
界面规格说明书 引入界面绑定时的级别	应用程序模型
	构架
更多详节	详细设计

更多级别上的用例	
完整的详细规格说明书 用例的最终级别	源代码

我无法在允许空间内涵盖所有相关主题，将在本文的余下部分作如下阐述：

- 需求应有层次地组织起来。
- 层次的分级不是功能的分解。
- 将业务需求与界面规格说明书分离开来。
- 不要从用例直接推论出设计。
- 对编写得当的用例有一组需求范围。

需求应该有层次地组织起来。这样做有许多原因。第一也是最重要的是，要管理需求的典型集合的复杂性，需要层次型的组织方式。人类处理复杂事物的能力非常有限。如果小组成员、客户及用户能够理解需求、需求存在的原因、调试它们，并利用需求验证开发产品，那么，需求必须是层次型的结构。我们的经验之谈是，系统的高层需求应该用不超过十二个左右的用例表示出来。在接下来次高的层，用例的数量不应超过当前层用例的 5 到 10 倍。

拥有不同细化级别用例的第二个原因是，对“测试”每一个领域、应用程序、构架以及详细设计模型而言，可以有完整的一组处于适当的细化级别的需求。这暗示着每一级别的“用例”必须在如下意义下完成，即在每一级别上覆盖所有类别的系统需求。第 N+1 级不应引入任何新的需求类别，但可把已存在的类别分解成子类别。如果在较低级别上，一个细化过的用例被发现不属于任何已存在的类别，所有较高级别的用例层次应该更新以反映新的类别。

用例最佳开发方式是迭代式和递增式的——与系统的其它可交付品相同。

层次的分级不是功能的分解。用例 1.1 不是用例 1 的第一步。用例 1.1 是一个特定的、更为细化的用例，隶属于用例 1 定义的用例类别。

将业务需求与界面规格说明书分离开来。在 Jacobson 论述用例的书中，使用了贩卖机的例子。其中描述的一个“用例”是，从贩卖机购买一项物品。“用例”示例细化了特定的接口机制，比如“把

硬币投入槽中”。这个示例没什么问题，只要用中立的术语描述过这一基本业务需求（如从客户处接受付款）。当第一级需求直接跳到接口规格说明书时，问题发生了。不幸的是，恰好大部分面向对象的项目小组认为他们就应该这样做。他们忽视了当以“用例”名义收集需求时的基本原则。这不能很好地服务项目。除非第一级需求是接口中立的，否则客户就被剥夺了考虑其它的可选对象的正常机会，设计人员也未被敦促构建系统的扩展性。相反，如果最初几个级别的需求是接口中立的，并且在显式的点引入接口绑定，用户对自己的真实自然的需求有更深入的理解，同时更能清晰地描述真实需求。例如，对下述情形，“存入硬币”仅仅是绑定到“接受付款”需求的一个接口，当它很明显时，促使软件设计人员、硬件工程师、市场人员、产品经理等等考虑其它接口绑定的可能性及暗含性，诸如一个电子货币或信用卡读取器。

不要从用例直接推论出设计。如果这样做，“用例开发”仅仅成为了功能分解的一个借口。用例止于系统接口的边界！用例应该描述参与者使用系统时所遵循的次序，但用例决不说明系统内部采用什么步骤来响应参与者的刺激。

软件系统是满足某一功能需求集的特定架构的实例。构架的选择或创建通常不是根据功能需求，而是缘于标准域关系及其它系统需求如时间、空间、可靠性、可分布性、可移植性、标准化，等等。我们刚从特定设计的黑暗年代浮现出来，进入基于可定制且标准的框架及模式创建构架的未来。我们不要踏出一大步，又回到基于功能分解一对一式的设计的年代。如果我发现另一个小组以“用例驱动方法”名义如此行事，真想在下届 Object Expo（对象博览会）时，由一支行刑队带着蒙着眼睛的整个小组去面对公众执行枪决！

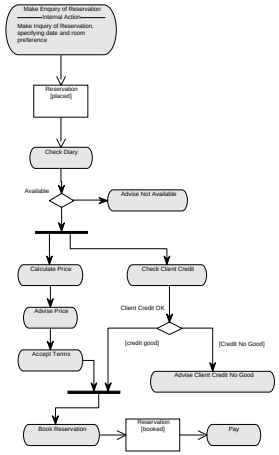
标准构架正被手工艺化，以满足非功能需求型的类的需要。构架被实例化以实现功能需求集的方式是采用一组对象交互图，而不是一堆用例。

对编写得当的用例有一组需求范围。一份免费的模板和白皮书可从我们的网页 www.software-architects.com 获取。

将需求组织成分散的用例是个好主意。我是用例的热情支持者，但常常是小组误用用例而让我沮丧。需求的收集过程是软件开发最困难也是最重要的部分。小心地管理它。

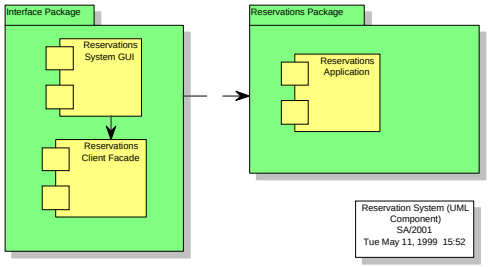
原文链接 http://www.umlchina.com/Indepth/Misuse.htm

UML Activity Diagrams（UML 活动图），可以完整地描述企业的工作流程，即人员完成工作所需经历的步骤，能补足 Class diagram 不足之处。 它最主要的价值是去发现所有的并行程序，以排除不必要的商业程序。

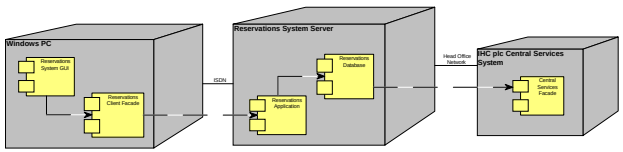


Implementation

A Component Diagram （构成图）用来塑造软体结构的模型，包含附属在软体间的元件、二进位码元件及可执行的元件。

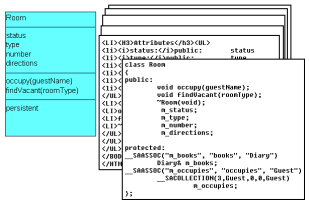


Deployment Diagrams （配置图）用来模拟处在执行状态的元素及其所涵盖的软体的元件、程序及物件的配置情况。 在这图型里，可以模拟实体结点 (Physical Nodes)及结点间沟通的联结关系。



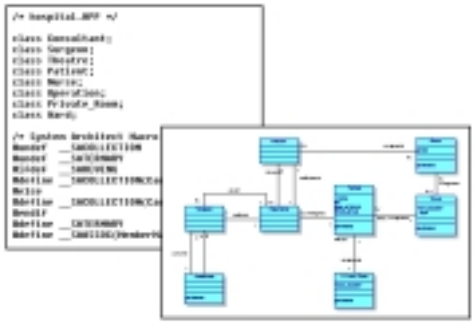
Generating Code

经由 Class Diagram 可以自动产生多种程式码；如：C++, Java, Visual Basic, HTML, CORBA IDL, Smalltalk, 及 Delphi Object Pascal.



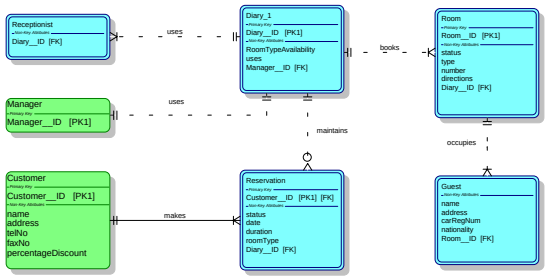
Reversing Code

System Architect 2001 可提供 C++ 及 Java 程式码的逆向工程。



Generating an ERD

您可以利用 UML Class diagram 自动产生 Entity Relation Diagram（实体关系图）。Classes （类别）的设计是与 Entity（实体）相对映。Class Attributes （类别属性）与实体属性相对映。



System Architect 2001

UML Object Oriented

物件导向系统

简 介

POPKIN SOFTWARE

戊丰科技股份有限公司
台北市敦化南路一段 21 号 4 楼之二
Tel：(02)2570-1877 Fax：(02)2570-1876

Methodology Outline

System Architect 2001 支援数种物件导向的方法论，包括： OMT/Rumbaugh, Booch, Shlaer/Mellor, and Coad/Yourdon，另也支援 Jacobson Use Case Diagram 以及 UML。

The **Unified Modeling Language** (UML).

Unified Modeling Language（统一塑模语言）是一个标示物件导向系统分析和设计的发展标准。 System Architect 2001 支援 UML 1.3 的规格。

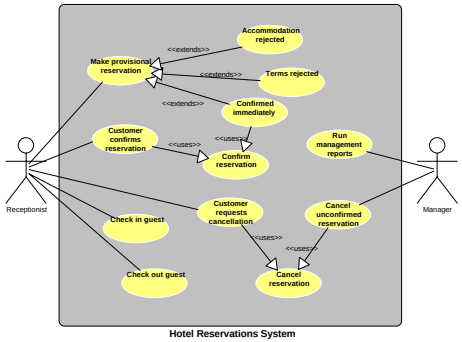
虽然 UML 并不是规定一个发展物件导向的标准程序或方法，但它是指定一个标准的图型及标记的组合，强迫使用者塑造确切的系统模型。

System Architect 2001 Features

- 32 位元使用者界面，及完整的企业模型整合工具。
- 多合一的电脑辅助设计工具（CASE tool ），包含商业描述、物件导向、元件设计及资料模型。
- 支援的法论包括： UML, IDEF0, IDEF3, IDEF1X, Yourdon, Ward-Mellor, Gane & Sarson, SSADM, Catalyst, OMT, Booch and Shlaer/Mellor
- 支援多人使用的网路版本，整合 Microsoft VBA、互动式关联矩阵(Matrix)及影响分析。
- 可以产生 Microsoft Word 文件及 HTML 网页文件。
- C++, Java, Dynasty & Magic – Generation & Reversing, HTML Generation, Simulation.

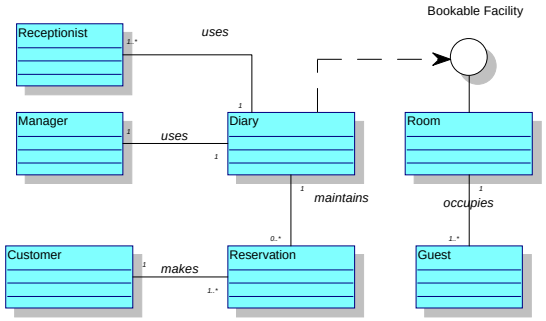
Business-Level Analysis

Use Case 技术可以获得记录该系统或企业目前所做的事及应该做的事**的资讯**。Use Case 技术可帮助您建立系统作业流程的架构模型。它是一个通向物件导向系统分析的绝佳方法。



Static Analysis & Design

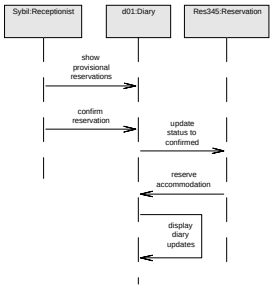
The Class Diagram 是一个系统主要的静态分析图，在这图型里，利用类别和继承的关系，释明了系统的类别结构。



System Architect 2001 支援物件间沟通的介面符号，但沟通介面无法执行(Implementation)，它们缺少属性(Attributes)、状态(States)或关联(Association)；只有操作的方法(Operations)。一个沟通介面（Interface）可能以印有"Interface"的矩形类别（Class）符号显示，或是以类似棒棒糖的符号来表示。

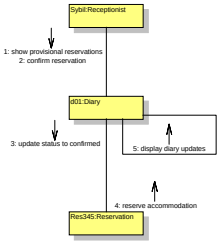
Dynamic Analysis & Design

UML Sequence Diagram 是用来塑造系统架构执行（Implementation）细节的模型。UML Sequence Diagram 表达出物件间交换的讯息(Message) 以产生期望的结果。



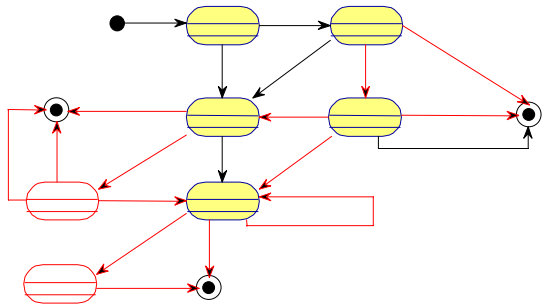
两个物件间沟通的讯息(Message)，由被呼叫的方法(Method Called)所定义；它是由传送的物件(Sending Object)指向接收物件(Receiving Object)；被呼叫的方法(Method)必须属于接收物件(Receiving Object)中所定义的类别实例(Instance of Class) 。

UML Collaboration Diagram （UML 联合图）是描绘物件间如何交互影响的模型架构。System Architect 2001 会自动同步给予 Collaboration Diagram 及 Sequence Diagram 相同的图型名称。



State Diagram（状态图） 是描绘系统运转的状态。 状态图是描绘系统内物件的每一个类别的典型， 包含重要的动态变化。

状态图（State Diagram）能利用父子图型的连结（Parent/Child Link）附属于类别图（Class Diagram）里的类别符号（Class Symbol）。



讨论请点击

构造用例过程：管理需求之二

Dr. Timothy Korson 著，June Fourteenth 译

虽然强烈地建议使用用例，但正如在上一期中所指出的那样，用例经常被滥用了。这次我将阐述一下误解和滥用最常发生的部分，即用例的**构造（Configuration）**过程。

我曾经为一家跨国公司开发一个分布式的，多币种和多语言的核心财务系统。该项目在许多方面都可作为当今面向对象软件开发的最佳应用范例。

通过和公司的注册会计师，核数员，出纳员，管理员以及其它财会人员进行广泛的接触，我用六周时间开发了一个全面的领域模型。这个模型包括一套完整的标准建模语言(UML)类图，共 17 张，存放在一个商用的计算机辅助软件工程(CASE)工具中。为了验证模型，还制做了一个遍历类图的视频，并分发给公司在世界各地的数百名担任重要职务的财会人员。制作视频需要一张放大的领域模型类图，这张图后来被贴在项目开发地点的墙上，而且经常被开发组用作参考，当然，有时也会修改。

整个软件的体系结构建立在基本的面向对象(OO)原则上(见图一)，并且重用了 Peter Heinckiens 在它的新书（Peter M. Heinckiens. BUILDING SCALABLE DATABASE APPLICATIONS, Addison-Wesley, MA, 1998）中详细描述的一个数据库的体系结构。这种结构在使商务逻辑完全独立于数据库技术的前提下，实现了由关系表到对象的映射。这使得移植到其它数据库相当容易。进行实际开发时使用了一个专业的 Java 集成开发环境，商业类库的使用也对开发成本得降低起了重要作用。在作重要的设计决定时我们会咨询领域问题专家，并且维护了一个技术问题解决方案的数据库，详细记录了正反不同方案的争论以及选择不同设计所依据的基本原理。

在项目计划中定义了 17 个系统增量。每一个增量完成时，都会在多个平台的多个数据库管理系统上进行测试。一个共同指导委员会被建立起来，以检查和协助项目开发。我们选择了一个开发人员和系统最终用户之间有着良好关系的地点进行β测试。当领域知识或应用需求中不明确的问题增加时，会有一个财务问题专家与开发组交流一两个小时。这一切都非常理想。财务人员对项目有着特定的兴趣，并且在作出决定之前他们可以发现所有的问题。

在进入系统开发后情况非常好，即使到最终我们也只定义了 18 个用例，它们之中没有任何一个

被详细描述到完整的交互对话框，也没有使用标准的 UML 详细用例模板。

我的许多同事认为这种缺少详细用例的方式是一种异端行为。我同意在大多数情况下应当付出更多的努力来开发用例，但是并非所有的项目都需要相同的用例构造过程。

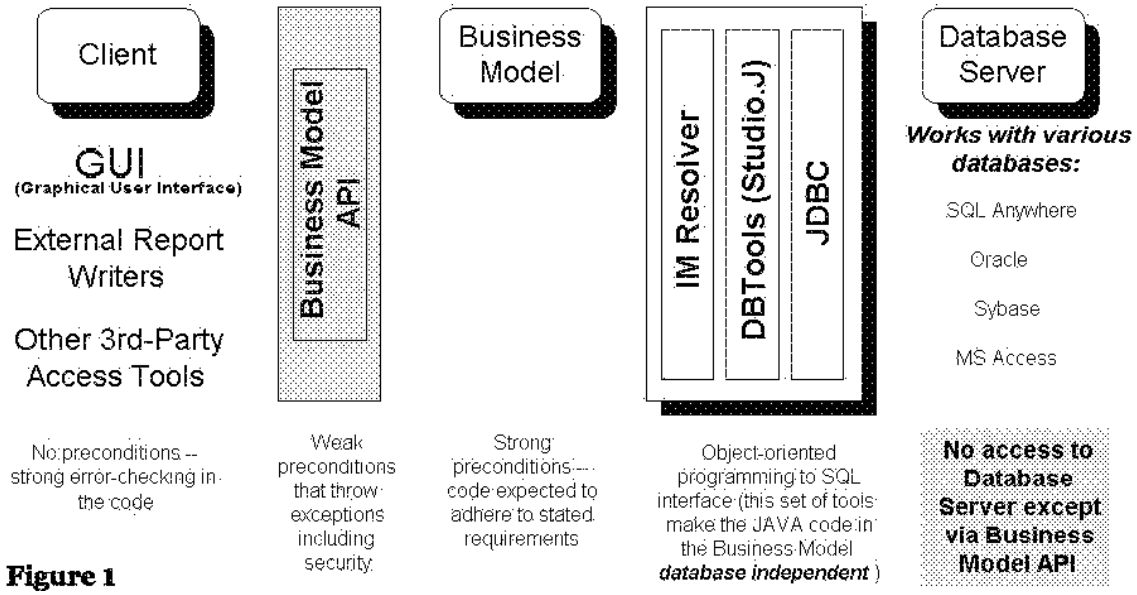


Figure 1

针对不同的项目应当构造不同的用例过程

一个完整的用例过程需要一组按层次组织的用例。这些用例应当包含充足的细节以得到最详细的系统测试样例。这些用例应当通过不同的方式， 在不同的前 提等等情况下进行完整的，详细的描述。每一个用例都应当通过顺序图(sequence diagram)来描绘其实现。

大型的，多地点的，多小组的项目通常需要完整的用例过程，但并非所有的项目都属于这一类。有几个因素影响到用例过程的构造。

多小组。项目组的成员越多，对用例过程的详细程度和形式的要求就越高。通常用例都是跨越组间的。编写良好的，详细的用例可以使小组间交流的特定方面文档化，这样可以避免许多误解。在一些地方提供咨询时，我们还扩展了用例模板和对应的顺序图，目的就是显式地表现出小组之间的边界。本文开头描述的财务系统是由一个单独的小组开发的，所有的小组成员都集中在一个特定的工作区域。

多地点。多地点的项目包括了多小组开发的问题，更加需要一份包括用例在内的，正式和详细的文档。

专门小组。我更喜欢“从摇篮到坟墓”的完整小组。但是许多公司由不同的小组负责需求分

析，设计开发和测试。当需求小组与开发小组分离时，很明显对文档细节的要求就增加了。上文提到的财务系统的开发小组非常完整，由同一个人负责编写 Java 代码，同会计师交流，建立领域模型以及测试系统。

小组成员的领域知识。当我在 Clemson 大学任教授时，曾在软件工程学院作过数年的访问学者。我总是想起在 SEI 月季系列研讨会中的一位特殊的发言者。他是做雷达信号处理工作的，他的论点是教雷达专家如何编程，比教程序员雷达技术要容易得多。他举出了几个有关的趣事来阐明自己的观点。他的大多数报告都难以反驳，这也是我坚决主张开发小组必须参与领域分析的原因。开发小组对领域知识了解得越少，用例模型需要提供的细节就越多。在财务系统的开发小组的例子中，小组的大多数成员都拥有会计学位和广泛的财务背景。

与系统测试过程的集成。如果用例模型与系统测试过程紧密集成，那么开发详细用例的支出就会在开发系统测试样例时得到补偿。如果一个独立的测试小组(他们必须用尽任何方法开发测试用例)在项目中及早地参与用例的开发，那么自然会有许多好处。在这种情况下，独立测试小组可以在项目的整个生命周期中参与领域模型验证和其它质量保证活动。对上文的财务系统来说，测试小组是不独立的。

需求已经以其它方式存在。如果一个大型的，多地点的项目，在开始时基本没有已有的书面需求，我建议需求收集过程应当包括开发一套完整的，详细的用例。如果项目是对现有系统的重新构造，或者由于其它原因已经有一套需求，那么小组应当决定：作为一套层次化结构的用例，应当重写多少比例的需求。影响决定的因素有很多。我建议至少建立领域用例最上端的两到三层，这样有助于驱动领域模型的验证。当需求已经以某种形式存在时，大多数小组会选择只开发用例的下面几层，通常会有 50 到 200 个用例。然而，在已有详细需求文档的情况下，每多一层用例，他们的附加价值会戏剧性地减少，却是一个合乎情理的事实。一个例外是用例过程紧密地与系统测试过程集成，并且独立测试小组直接由详细的用例来构造测试样例的情况。上文的财务系统的需求已经在几本大部头的手册和遗留系统中存在了。

需求的稳定性和客户信息的可用性。当需求改变时，用例必须保持精确和更新。如果需求相当稳定，维护的任务就容易管理。然而，如果需求迅速地变化，更新数千个用例的可能会难以接受的高昂。在某些情况下，当客户信息容易得到并且反应迅速时，小组可以选择放弃书写详细的接口规范，而代之以在实际的项目进行中不断的请客户参与审查的过程。财务系统的接口规范并未很好地定义，但客户总是乐于同我们合作。

还有很多其它的因素，包括应用本身的特点，测试自动化的程度，资源的可用性，市场的时间限制，公司全体文化等等，都会影响用例过程的构造，尽管大多数项目经理认为如果要使用对象，就有且只有一种正确的方法来使用用例，我仍然要强调的基本一点是，一个尺码不会适合所有人，

经验使我相信，在用例的谱图中(见图 2)。项目总是处在某个中间位置，灵活地构造你的用例过

程，并且仔细地管理它们。

图 2

重量级用例过程	<	>	**轻量级用例过程**
多地点			单地点
多小组			单开发小组
开发小组的领域知识有限			开发小组的领域知识广泛
集成的测试			没有外部的测试小组
缺少已有成文需求 稳定的需求规范 和客户缺少接触			丰富的已有成文需求 快速变化的接口规范 和精通业务的客户及时接触

原文链接 http://www.umlchina.com/zippdf/usecase2.zip

KCOM 技术介绍

KCOM 是 KCOM 公司独立开发的一系列开发平台和计算机语言技术的总称。
它包括：KCOM 组件模型，KCOM Basic 语言，KCOM Stage 运行环境，KCOM Space 开发平台四项技术。

KCOM 组件标准：
KCOM 组件标准是一个完善的组件标准，可以自定义属性，方法和事件，并使用事件驱动的方式编程。同时源代码方式存储有利于在网络上传输。

KCOM BASIC：
语法结构完善，易学，可以使用中文进行编程，例如使用中文变量名称，中文方法名称，属性名称等等。独有组件运算功能能够实现代码的可视化。

KCOM Stage：
KCOM Stage 是 KCOM 组件的解释器，由于采用了代码的结构化存储，在解释之前不需要对语句进行扫描和文法处理，同时采用了高效的寻址方式，所以比一般的脚本语言的解释效率要高。

KCOM Space：
能够方便的编写 KCOM 组件和代码，是一个可视化开发平台，具有丰富的界面排版，代码编写和代码调试功能。

KCOM Space 以其自定义的组件标准(KCOM 组件标准)为核心，完整地实现了一门计算机语言（KCOM Basic)、一个完整的组件化开发平台（KCOM Space）和一个高效的虚拟机（KCOM Stage)。作为国产软件在高端领域的突破，KCOM 表现出了与世界同步的技术水准。由于采用了全面组件化的思想，抛弃了传统的设计方法，全面革新了开发平台从底层的语言到上层的操作界面的设计。

KCOM 公司的产品在 2000 年中国国际软件博览会上获得创新奖。并受到倪光南院士的高度评价。

KCOM 公司合作意向：

KCOM 公司在长期对于开发平台和计算机语言技术的关注与不断开发的基础上，形成了自己独特的技术特长与优势。在如今的软件领域，软件和开发平台日益紧密成为趋势，其中突出地表现在：

软件与基于该软件的二次开发平台：如 MS Office，Notes，GIS 应用软件及开发平台，3D Max，AutoCAD，Director 等等应用软件。 一些稍小的软件如 InstallShield 也在其中使用了自己定义的解释语言，以提供强大的扩展功能。

企业流程快速建模与软件生成工具：此类工具在国外的企业应用解决方案市场中被大量使用。而国内的系统集成企业正在开始开发和形成自己的解决方案与定制工具。

嵌入式企业应用前端设备：该设备是我们所看到的一个巨大的市场机会，基本原理在于软硬件一体的显示与操作前端，采用 NC 体系，运行企业应用。

在以上几个领域中，KCOM 公司都能够以自己长期的积累和技术特长，与合作方展开广泛的技术合作，希望能够来信探讨：zhangiii@163.net

http://www.kcomspace.com.cn/

讨论请点击

创建有用的用例：管理需求之三

Dr. Timothy Korson 著，June Fourteenth 译

“软件工程师毫无用处，我宁愿雇佣雷达专家教会他如何编程，也不愿雇佣程序员来教会他雷达信号流程”。这段措辞强硬的声明来自于一位大型政府项目的经理，他部门的电脑系统曾错误的发出了 ICBM 导弹来袭的警报。更加令他不能忍受的是程序员拒绝对此项错误的功能承担责任。程序员抱怨说这是由于系统说明不全造成的，而非他的原因。这位经理意识到，需求文档没有指出特定的环境会导致错误的警报，但他仍认为在他部门的编程人员应该具备可以解读这一切的基本知识。 “没有任何雷达专家会犯如此基本的错位”，他坚持说。

就我的经验来讲，技术并非系统开发中的主要问题。最为重要的是怎样处理需求--- 如何得到正确的需求，以及如何正确的获得需求。系统越大越复杂，需求问题就越成为风险最高的因素。无论在怎样的一个领域里这都是千真万确的。我曾亲自在航天，电话，金融系统，仓储控制这些完全不同的领域中发现过同样的结果。

本章的主要观点是简单的运用用例来获取需求并不意味着你能得到完善的需求。用例只是提供了收集需求的框架和表达需求的方法，仅仅一个好的格式是不够的。鉴别一系列完全的角色意味着我必须捕获所有用户的观点，但是如果某一角色并不真正了解商务需求会如何？如果开发人员错误的理解了用例又将如何呢？这两种情况都将导致开发出的系统无法满足用户的要求。

我认为建立一个好的用例依赖于在应用领域的分析中所获取的知识。反过来，知道不同的角色和他们在这一领域中的主要行为对了解一个领域也有帮助。这表明迭代的进行领域分析和用例的建立是获取好的需求的首要条件。我将在以后的文章中详细讨论如何进行领域的分析。简单的说领域的分析就是就将领域的概念，这些概念的特征和行为以及各概念间的关系记录下来的开发过程。UML 是捕获领域模型的标准符号。必须注意的是用例的建立是定位于功能，而领域的建立是定位于概念的。一个领域模型并没有软件系统的符号，领域模型中的类是纯粹的领域的概念。

让我惊奇的是许多项目都跳过了领域分析这部分。我看过许多项目有详细的用例，和设计的各种图形，但没有领域模型。我愿意打赌在这种情况下开发人员无法很好的理解用例，当然并非说设计不重要。

如果不了解这一领域，你就无法创建正确有用的用例。 这对于客户和开发小组同样适用。别以为客户能知道并清楚的说明真正的商务需求。很多时候当你已一一实现了客户的需求说明但他们仍不喜欢你的产品，因为他们的说明并未真正反映他们的需求。在领域分析的过程中，客户必须要阐述他们对这一领域的理解。领域分析需要从几个方面合成这种理解。很典型的，每一个角色和这一领域的专家对系统涉及的这一领域的观点都是有限的。比方说在医院里，护士，医生，实验员，管理员都有着不同的观点。在每一种人中挑选一位作为代表参与对概念的讨论与建模可以增加每个人的理解。我们发现，有了多种角色，客户和领域专家参与到 UML 领域分析的类图创建中来，开发小组会发现他们需要重新更改以前对用例的定义。当领域模型接近完成的时候，我们会常听到客户说：“我以前从未对这一切有如此透彻的了解。”许多时候我们发现经过领域分析后需求文档需要重写。

如果你不了解一个领域，你就无法将一个正确的用例正确的实现。让我们再来讲一个例子，有一个财务的项目，用例要求会计能打印一份分类帐清单。单词 “分类帐”对会计人员来说有好几种不同的意思。他可以是一个原始交易记录的帐本；可以只是一组交易的记录；也可以仅仅是交易种类的参考记录(如现金，销售)。没有对会计知识的详细了解，软件工程师很容易错误的理解描述正确的用例。

你不可能一开始就能创建完全正确的用例，因为你对领域的理解会随着项目的开发而日趋成熟。同样的，我们对于真正的需求的理解也会随着领域分析的进行而增加。而一个人对领域的理解也会随着系统的设计和实行的过程而增加。那就是说要找到有用的用例，项目的所有阶段都要进行迭代/递增的开发方式。我看到过很多项目经理，他们认为他们的项目已经进行了很好的迭代/递增，但实际上只有细节设计和实现进行了迭代。一个好的项目管理应该有计划的再次进行用例开发，领域分析，架构设计以及细节设计和实现。

在第一次领域建模完成以前，不要写太多的用例。在领域的建模期间，这一领域的专业词汇被一一解释。最好别在基本概念清楚前对功能需求钻研太深。我建议在领域分析开始前鉴别出所有的角色和对应于每个角色的少量的用例，不要进行的太深入。如果系统有其他的限制和规定，尽量拿到一套完整，详尽的资料，并准备好在开发过程中更改需求。

顶层用例需要描述 “为何”创建。 说明书总是不完全的，部分原因是人类的语言不够精确，部分因为用例作者以为读者已经了解了领域的一些知识。我将在以后的文章中详细的讨论和举例。

好的软件工程不是用例驱动的。需求是重要的，用例是构造需求的好方法。但如果你同时要考虑开发的所有因素包括重用，架构，花费，时间，你就无法仅仅从一个方面来驱动你的项目。Ivar Jacobson 是我的朋友，我很崇敬他在技术上的专业，也在他的文章中找到了好的建议。然而我相信很多从业人员误解了他的建议。我以前也曾经说过软件工程是领域类模型概念驱动，但这和说软件工程是用例驱动的一样错误。好的软件工程是被一系列重要因素所驱动的，而且因素也因不同的公司和项目有着不同的重要程度。这些因素包括：技术上对于设计的考虑，用户需求，重用，可更改性，系统性能，标准化，日程的安排以及其他的商务驱动。每个项目都有着自己不同的考虑。对于每一种情况，可以精确的说项目

被域模型和用例共同驱动。

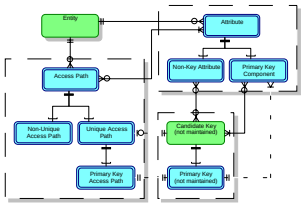
那些不关心领域的开发人员是完全没用的人。 这是从以上的讨论得出的结论，我只想在此把它清楚明白的讲出来。

我并不是说所有开发人员需要变成领域专家或者具有和他们的开发技能一样水平的领域分析技巧。技术专家非常珍贵。我想说的是这之间的区别已不存在了。持久的团队才是高产的软件开发组织。在一个小组里，我希望每个人都有专长，但我也希望每个人都能在某种程度上参加这些阶段。特别是我希望每个人都能理解和描述领域模型的细节。这虽然不能使开发人员对每一个用例都有正确的理解，但至少有很大的帮助。

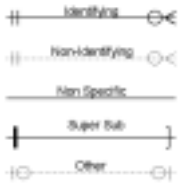
原文链接 http://www.umlchina.com/zippdf/usecase3.zip

Access Paths

Access Paths specify the attributes that can be used to locate an individual record that is represented by an entity. Access paths correspond to indexes in the physical data model.



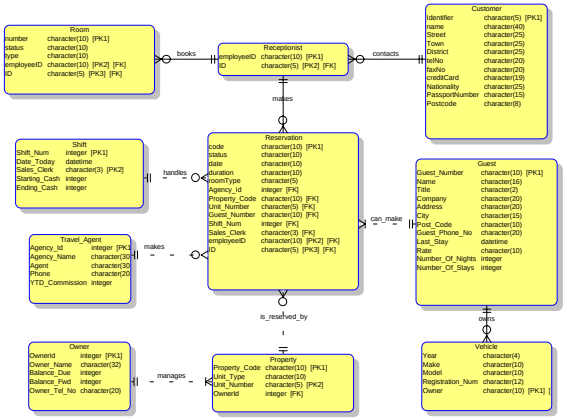
Relation Lines



Relation lines 表示实体间的关系，可能是父 / 子关系（parent/child relationship）、 Super-sub types 或 non-specific。 The nature of the parent/child relationship can be further categorised as Identifying or Non-Identifying.

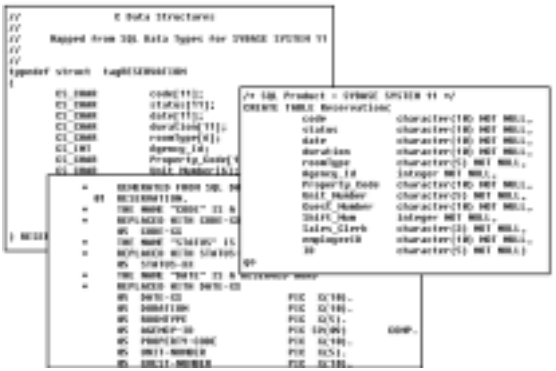
Physical Data Models

实体资料模型（**PDM, Physical Data Model**） 是用来描述一个特定的资料库，因此，实体资料模型不同于实体关系图，它只建构一个已完成的资料库模型。实体资料模型（**PDM**）可以直接转换成实体关系图（**ER Diagram**），相同地，实体关系图 **ER Diagram**）也可以直接转换成实体资料模型（**PDM**）。



Schema Generation

SA Schema Generator 准予您由实体资料模型（PDM）产生资料库架构，并且支援最普遍使用的资料库管理系统(DBMS)，SA Schema Generator 能输入至一个档案，或这个资料库能经由现存的 ODBC 做修改。

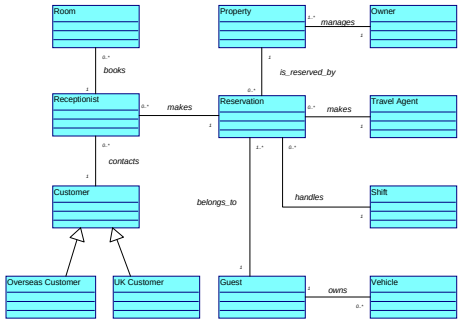


Reverse Data Engineer

资料逆向工程（Reverse Data Engineering），能让您将已存在的资料库结构转换成实体资料模型**(PDM)**。转换介面也提供第四代程式语言(4GL)应用程式，例如：Visual Basic, PowerBuilder, Delphi and Magic。

ER Diagram to OO Model

您能由实体关系图**(ER Diagram)**自动产生一个 UML Class diagram ;实体(Entities)对映到类别(Classes)，属性(Attributes)对映到类别属性(Class Attributes)。



System Architect 2001

Data Modelling

Quick Reference

POPKIN SOFTWARE

戊丰科技股份有限公司
台北市敦化南路一段 21 号 4 楼之二
Tel：(02)2570-1877 / Fax：(02)2570-1876

Methodology Outline

System Architect 2001 提供对商业模型、物件及元件模型、关联性资料模型及结构性分析及设计等四个面向的整合性支援。

Data modeling 包含实体关系模型（ Entity Relation models with subject-areas）、独立的实体模型（physical models）、 schema generation 以及资料逆向工程（reverse data engineering）。

System Architect 2001 provides the capability to build multiple project data models in a project encyclopedia. Project Data Model 能由实体关系模型图(Entity Relation Model Diagram) 及一个或多个 Subject Area Diagrams 所构成。图型及关系的定义显示出一个资料库系统，它可能是一个单一的资料库或一个分散式资料库系统。

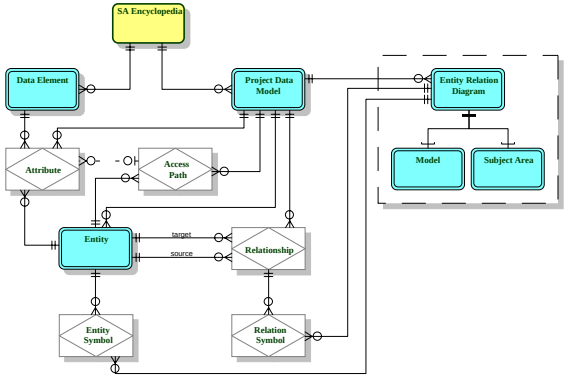
The Model Diagram 包含 project data model 内所有的实体及关系。The Model Diagram is built upon subordinate Subject Area diagrams. 每一个 Subject Area diagram 表示 Model Diagram 的一部份。

System Architect 2001 Features

- 32 位元使用者界面，及完整的企业模型整合工具。
- 多合一的电脑辅助设计工具（CASE tool ），包含商业描述、物件导向、元件设计及资料模型。
- 支援的法论包括： UML, IDEF0, IDEF3, IDEF1X, Yourdon, Ward-Mellor, Gane & Sarson, SSADM, Catalyst, OMT, Booch and Shlaer/Mellor
- 支援多人使用的网路版本，整合 Microsoft VBA、互动式关联矩阵(Matrix)及影响分析。
- 可以产生 Microsoft Word 文件及 HTML 网页文件。
- C++, Java, Dynasty & Magic – Generation & Reversing, HTML Generation, Simulation.

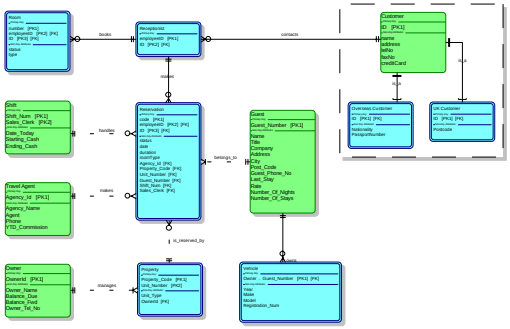
Project Data Model

Project Data Model 能由实体关系模型图(Entity Relation Model Diagram) 及一个或多个 Subject Area Diagrams 所构成。



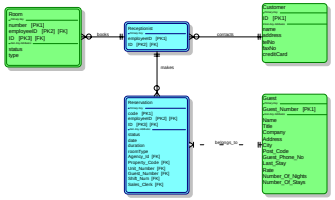
Model Diagram

The Model Diagram is built upon subordinate Subject Area diagrams. Subject Area diagram 的改变，会自动反映在 Model Diagram。



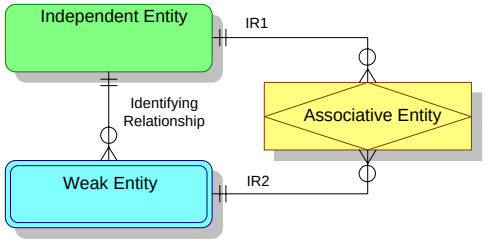
Subject Area Diagram

每一个 Subject Area diagram 表示 Model Diagram 的一部份。



Entity Symbols

实体（**Entity**）表示您企业组织内用来记录的资料内容，包含人、事、物、概念等实体。System Architect 2001 提供所有的实体符号(Entity Symbols)。

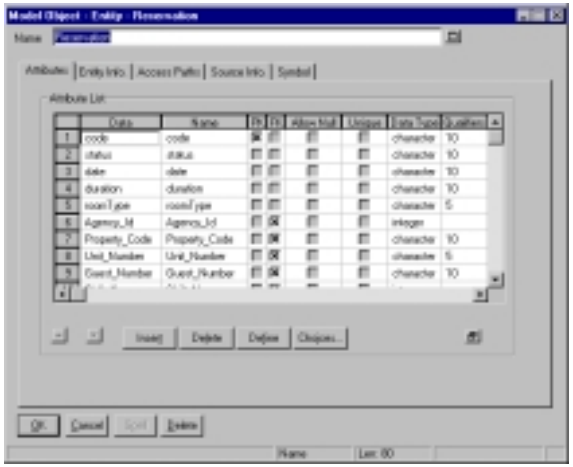


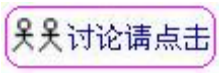
System Architect 2001 自动改变实体符号，以反映出适当的从属关系。

Entity Attributes

属性（**Attribute**）是定义实体（**Entity**）与资料项目（**Data Item**）间的关系。

The Attributes grid 准许您定义目前的实体及资料项目属性，也能被重复使用在其他的 SA design activities, process modelling, Data Flow diagrams.





《分析模式：可重用对象模型》

学习笔记之四：企业财务分析中的观察和测量

Windy J

这一章的内容基于第三章的观察和测量模式，所以，请首先阅读第三章的内容。

在大规模的企业中，高层的问题很容易被发现，但是要发现这些问题的根源却比较困难，因为这样的企业会产生大量的信息，以至于很容易分析人员就被淹没在这些信息当中了。

例如，一个企业最主要的财务表现是它的最终财政收入，如果财政收入有问题，就必须找出引起问题的原因。在 ACM（Aroma Coffee Makers）的财政收入中表明他们的设备销售收入减少了，尽管花费还同样可观。这个问题在 ACM 的东北区域最为严重；继续深入追究发现他们的 11－10 单元低于计划收入，尤其是面向政府的部门，到这里为止，这些分析的大部分仍然是数据的形式；进一步分析可能给出定性的结果：可能是由于无力的销售补偿计划，或政府预算缩减，或炎热的夏季，或出现了强劲的竞争对手，等等。

这样的分析过程非常类似于医生对病人症状的分析：根据显著的外在症状，用专业知识跟踪可能的病因，然后对症下药。这种类似性让我们意识到，噢，也许观察和测量模式可以应用到金融分析中来。让人高兴的是，确实如此，那些模式非常适用，只需要做一些适当的调整，而且调整基本上是对已有模型的扩展，而不是改变。

下面我们就来看这些模式和它们新的应用。

1.1 企业部门单元（Enterprise Segment[1]）

首先我们要做的小小补充是引入 Object of Care 来表示被观察和测量的对象，在前一章中，观测的对象是 Patient，而我们现在关心的对象是企业的部门，所以增加一个抽象层次，让 Patient 和 Enterprise Segment

[1] Segment：本意是片断的意思。

都成为它的子类；我们顺便介绍一个 Population 类（人口）来代表“人”的群体，我们知道，对于人口而言，也有很多有意义的观测工作，这是题外的话。引入的抽象关系请看下图所示：

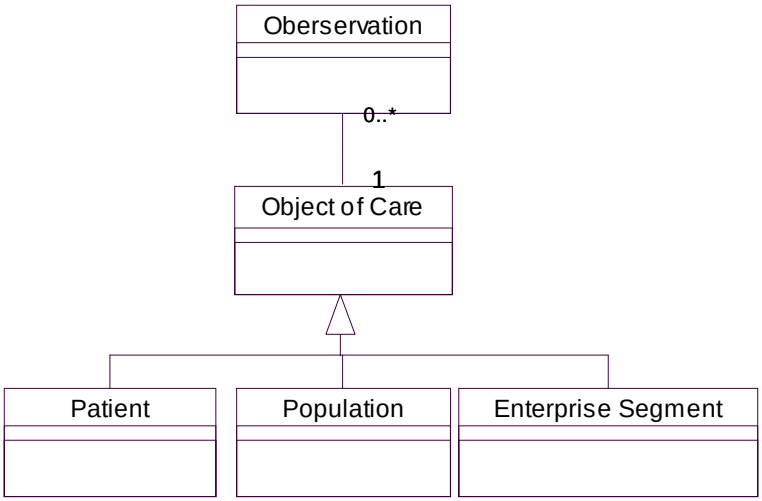


图 1 Object of Care 和它的子类

在这里还要提一下 Enterprise Segment（企业部门单元）的来历，当我们面对一个企业时，总是可以根据一些原则来把它划分成不同的单元/部门，根据地理位置，根据产品，根据产品销售的目标行业，等等。例如一家国际性的公司可以首先根据市场划分（例如美国），然后根据区域（Region）划分（例如东北地区），然后根据地区（Area）划分（例如新汉普）。这些独立的层次中每一个方面都是一个维数（Dimension，即市场，区域，等）；而新汉普和东北地区是“地理位置”维数中不同层次的两个元素；一个 Enterprise Segment（企业部门单元）是各维数元素的结合，即取每个维数的一个元素构成，例如 ACM 的一个部门可以这样描述：东北，11－10，政府部门；东北是地理位置，11－10 是产品型号，政府部门是销售对象。

通过这样定义 Enterprise Segment（企业部门单元），我们得到一个企业结构的描述模型，如图所示（见下页）：

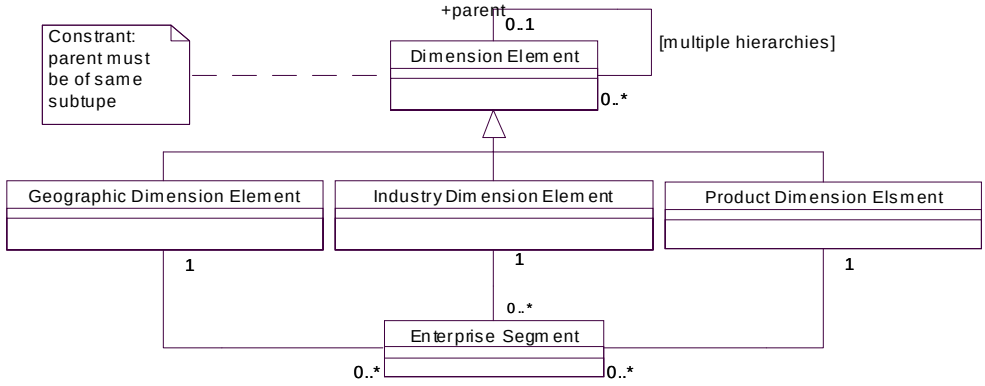
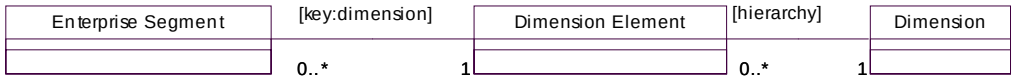


图 2 用维数元素定义企业部门单元

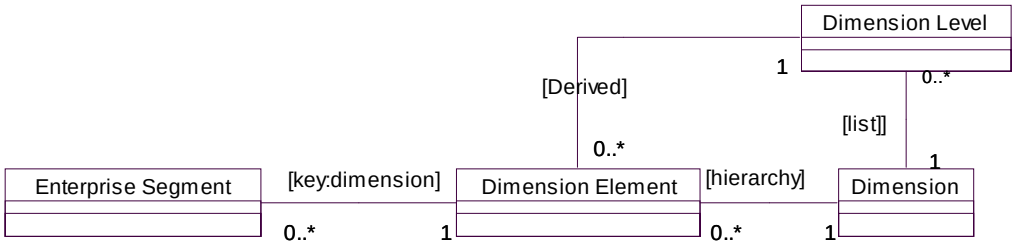
这个模型基本解决问题，可以方便定义层次，以及反映了三个不同的维数元素，也反映了企业部门单元和维数元素的关系，但是很快可以看到，在这里，维数被局限在三个：也就是说，如果维数需要改变，那么整个模型都需要更改，那是我们不希望看到的。因此稍作改进之后，得到以下一个模型：



在这个模型里，增加维数变得容易了，而且，用关键字映射（keyed mapping）以及数量上的约束来保证 Enterprise Segment 在每个维数上仅对应一个 Dimension Element。

每一个层次都需要一个顶层元素，可以定义为“all”，或者“空（nil）”。

如果考虑到维数级别（Level），可以修改上面的模型如下：



在这里，可以定义每一个维数元素的级别，而级别由这个元素在维数层次中的位置决定的，例如，在维数层次中，元素新汉普的上级是东北地区，再上级是美国，往上是 all，所以它的级别为 3，得到维数级别在 list 中对应的结果是地区（Area）。

1.1.1 定义维数（Defining the Dimensions）

要得到比较合理的 Enterprise Segment（企业部门单元）定义，维数（Dimension）的定义也是很重要的一个环节。最简单的方法是遵循某种显而易见的组织结构，但是不同的分析会有不同的侧重点，所以这样也并不是每次都可行。

一个更好的办法是观察层次最底层的部分，然后看在那里的是怎么分类的，例如在 ACM 的例子里，可以看到重点是咖啡机的出售和出租，因此可以得到咖啡机类别，出售出租地区和销售目标行业这三个维数，用一个名词焦点事件（focal event）来表示这种分类的依据。

分类可能会很复杂而出现交叉，但是，在现实的情况中，很少出现维数（Dimension）会大于 6 的情况，导致实际分析中可能需要对上一节的模型稍作修改：尽量考虑一下有没有这样的必要。

维数（Dimension）通常不必细分到最末端的一层，例如在 ACM 的例子里不值得，或者甚至可能一直往下分析到每个销售人员的销售范围甚至细到每个具体的顾客，这样的层次对理解整个系统结构是有帮助的，由于复杂度以及具体细节暂时不进行深入讨论。

维数可以由系统分析人员显式定义；否则，就得由企业数据库决定，如果是这样，每个维数就得定义一个 builder 操作从数据库查询所要的数据，这也允许系统随着时间的推进增加维数的节点（add nodes to the dimension over time）。

1.1.2 维数和企业部门单元的属性（Properties）

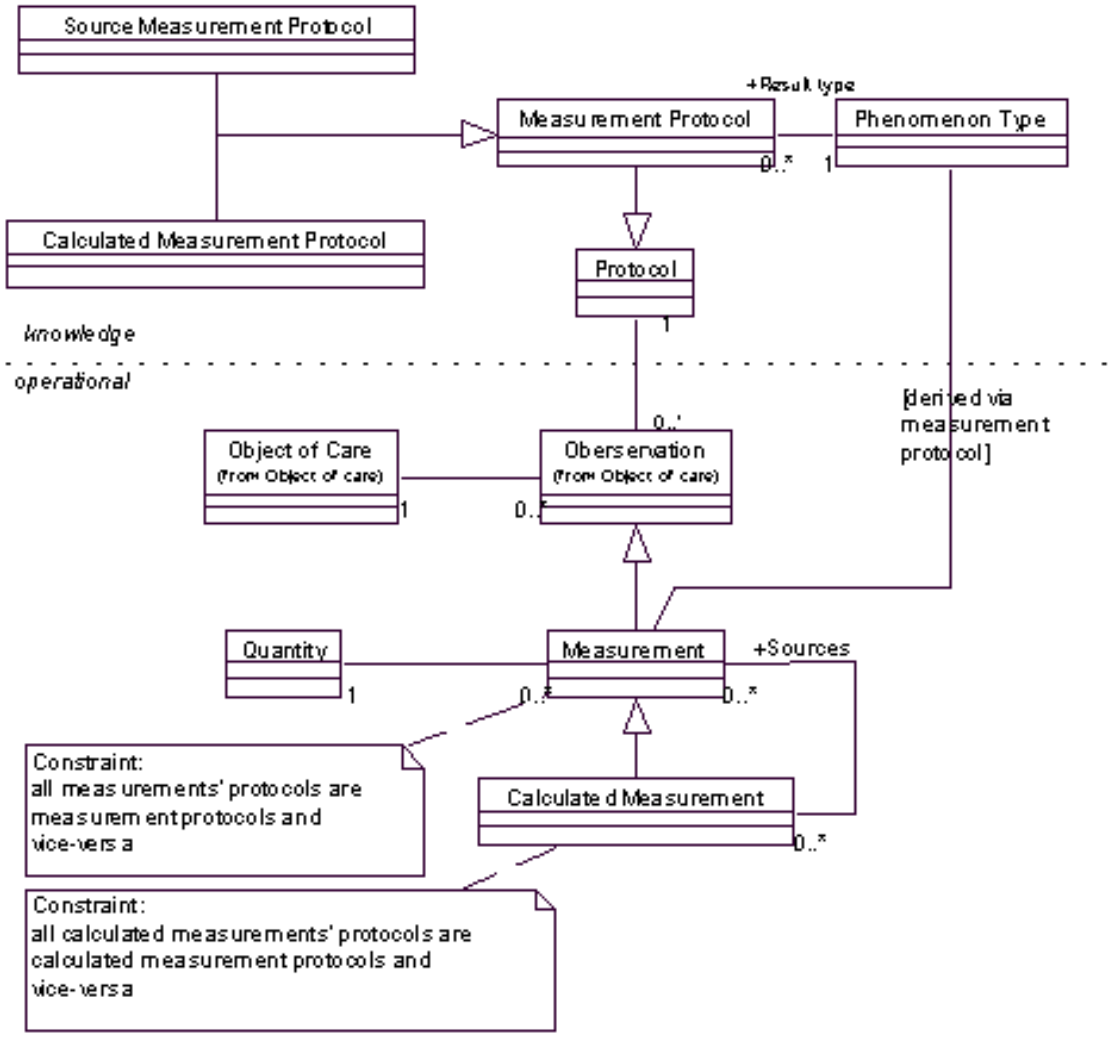
对于维数（Dimension）而言，一条非常重要的规则是维数层次中，上面级别的测量可以由底下级别的测量得到，例如我们想得到东北地区的销售收入，就可以累计东北地区下属地区所有的销售收入。每一个维数都必须支持这一规则，具体运算通常是累加，但也有例外（参见本文第 2.5 节）。

维数的定义经常跟企业结构有关，不过还有一个非常重要的通用维数是时间（time），也满足上一规则。

企业部门单元（Enterprise Segment）还有一个有趣的属性，那就是：它们是概念存在的，就像许多基础的数据类型一样，哪怕它们并没有创建成为软件对象。我们仍然把它当作非基础数据类型，创建时是一种查找/创建形式，即首先查看要求的实例是否存在，如存在，返回它作为结果，如不存在，创建它。

1.2 测量协议（Mesurement Protocol）

在企业金融分析中要用到许许多多的测量（measurements）,这些测量不是用手工输入的，它们通常来自数据库，或者从其他的 measurements 计算而来，也就是说，获得这些测量的途径很重要，下面的模型给出了关于测量协议（Mesurement Protocol）的一个大体框架，跟第三章中的模型很类似：



在上述模型中，给出了两种测量协议：Source Measurement Protocol 和 Calculated Measurement Protocol，其中 Source Measurement Protocol 指的是从企业数据库查询，一般来说一个对象知道去哪里取数据，哪怕实际的代码可能在另外的层次中（用户应该决定）。而 Calculated Measurement Protocol 代表测量由系统中已有的测量对象计算而来。

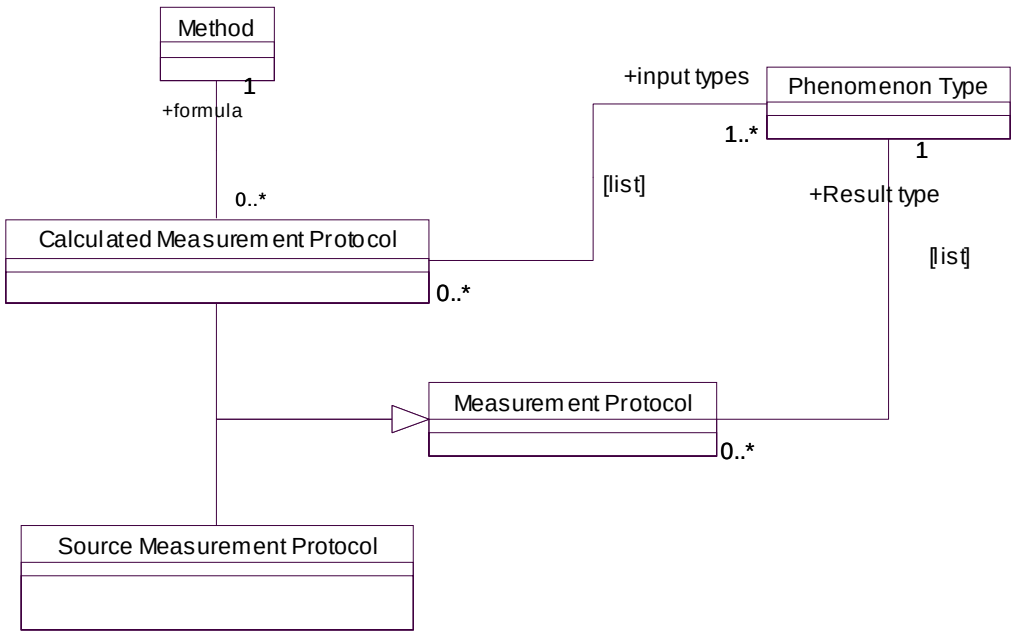
需要指出的是，在这个模型中，一种类型（Phenomenon Type）可能对应几种测量协议，例如，同一个 Phenomenon Type 可能同时拥有源/计算协议。一般来说应该由用户确定到底采用哪个具体的协议，但也

可以从 Phenomenon Type 到 Measurement Protocol 形成一个映射表，并从前到后表明各个 Measurement Protocol 的优先级，以供系统选用。

还有要注意的是 Calculated Measurement，连接到它的源（Sources）测量对象，这是基于计算结果的通用原则：当结果作为对象时，它应该知道哪种计算方式给出结果（协议），还有对这个协议的输入（Sources）。

1.2.1 保存计算结果（Holding the Calculations）

Calculated Measurement Protocols 包括计算的公式（formulas），这些公式一般非常简单，所以用简单的解释器（interpreter2），并在解释器中把公式用一般电子表格格式来保存。由下图可以看到：



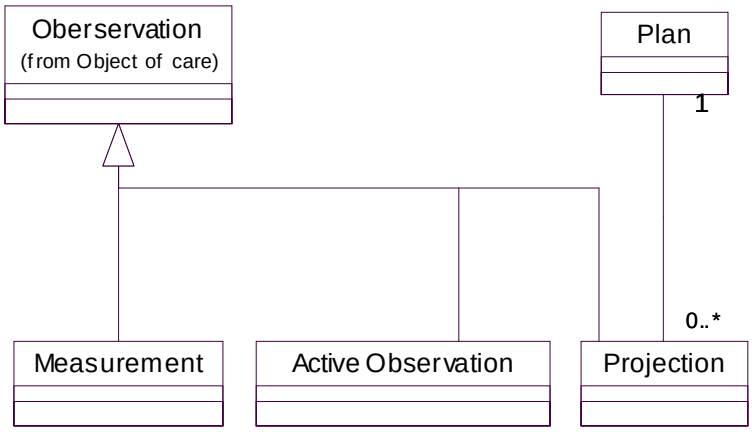
注意图中参数的表示方法，每个 Calculated Measurement Protocol 有一个参数列表，该列表同公式中提到的类型相结合，由于这些类型对象必须可识别，所以用列表是合适的；也可以用字符串关键字表示。

1.2.2 比较关系和因果关系的测量协议

在企业金融分析中，测量（Measurements）并不都是数值（value），例如收入多少多少美元，人们可能对比较实际收入和计划收入，或是对目前数值和历史数值对比更感兴趣。考虑到这些比较性的测量，我

[2] 参见《设计模式》一书相关模式。

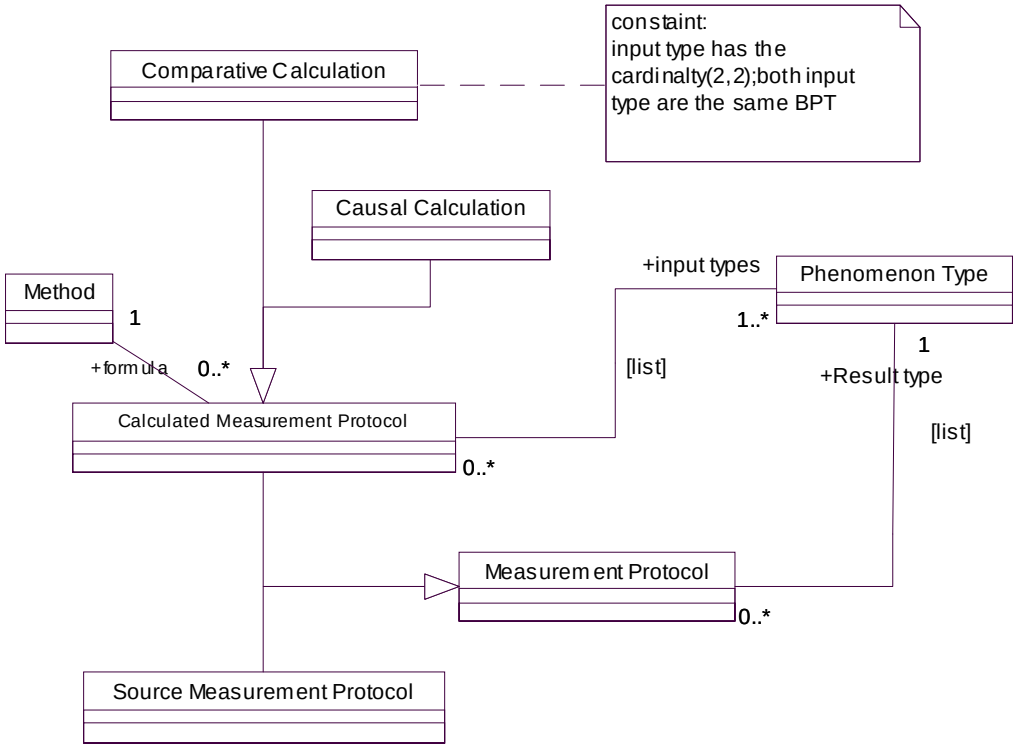
们需要在模型中加入它们，鉴于一般的比较都是在历史数据和现实数据，或者计划数据之间发生，而历史数据可以由计算某个时间段得到，那么计划数据稍有不同，在上一章中提到过 Projected Observation，在这里，Projected Observation 还要和相应的计划结合在一起，如图所示：



需要注意的是，计算测量（Calculated Measurement）可以分为两大类，一类是因果关系的测量（Causal Measurement），因为这类测量的结果来自其他的测量（计算之后），因果测量可以有不受限制的输入参数，输入之间任意的关系，而且计算公式也可以是任意的形式。

另一类就是刚才说的，比较关系的测量，相比而言，这类测量要有规律得多，一般是两个输入参数，必须是同一种类型（Phenomenon Type），比较结果要不是两者之间的差值，要不是差值对于输入的百分比，即：（x-y）或（（x-y）/y *100%）。

这两种不同的测量协议可以由以下的模型来描述：



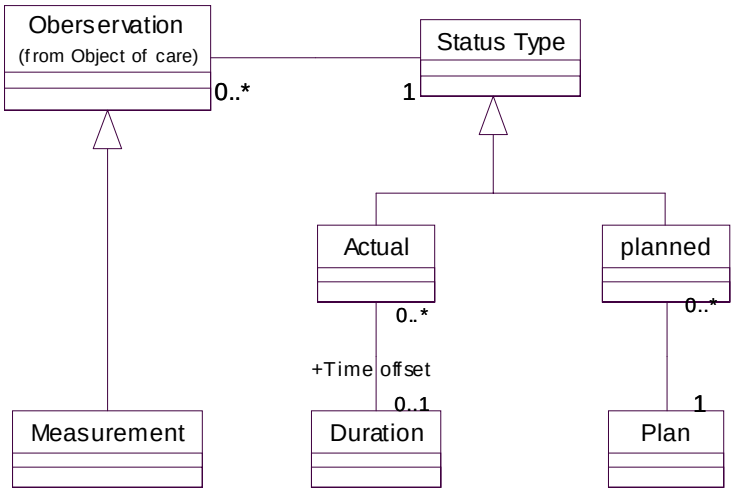
可以看到，在 Comparative Calculation 中，有对输入参数类型和数量上的限制。

1.2.3 状态类型（Status Type）：定义计划（Planned）状态和实际（Actual）状态

由数据源或计算所得的测量（Measurement）通常通过他们的测量协议得到结果，测量协议（Measurement Protocol）可以为测量（Measurement）提供一个工厂（Factory3）方法（来创建测量对象），当客户需要创建测量对象时，需要提供测量的对象（Object of Care），需要说明是实际测量还是计划测量，如果是计划测量，则需要给出相关的计划，如果是实际测量，则需要制定实际测量的日期。

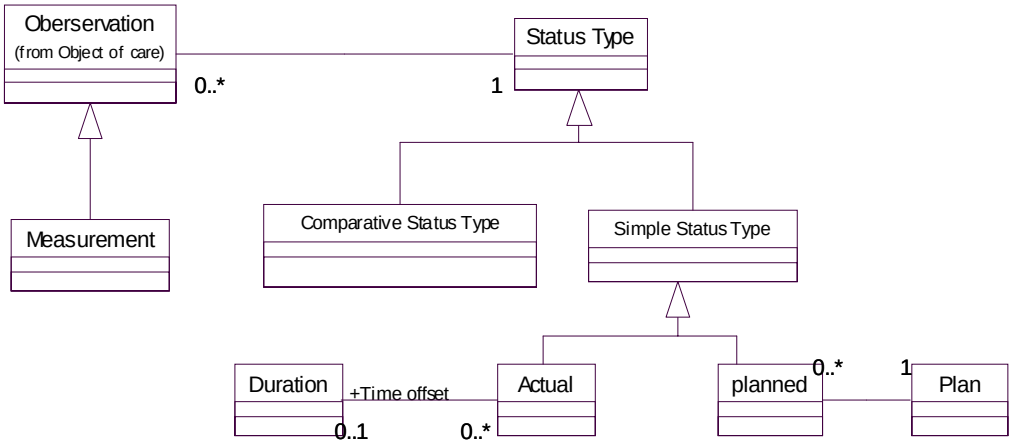
想想，是不是可以把测量的状态信息提取出来到 Status Type 类，单独表示？如下图：

[3] 参见《设计模式》中的 Factory method。



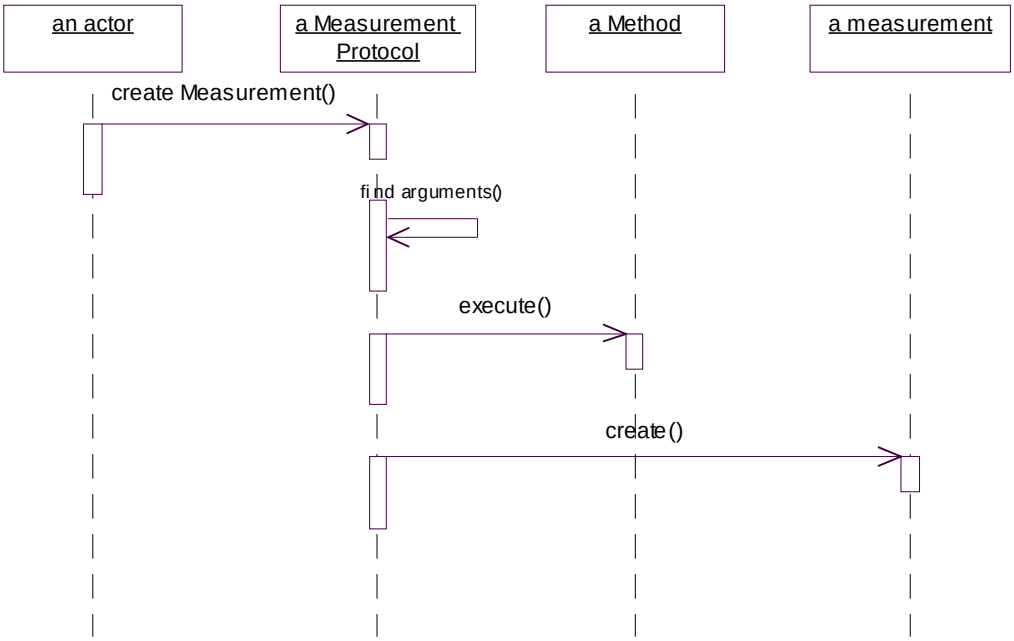
在这个模型中，把观察的状态从 Observation 当中抽取出来，放到一个单独的类中间，这样还可以方便扩展到其他可能的状态。而且，客户创建测量时只需指定测量状态就可以了，如果是比较性的测量，则需要两个状态，给予两个参数。如果对这种情况更进一步明确表述，还可以得到下面的模型：

疑问：原文中 Actual 和 Planned 是两个 Status Type 的子类，而笔者认为，作为 Simple Status Type 的子类就可以了。

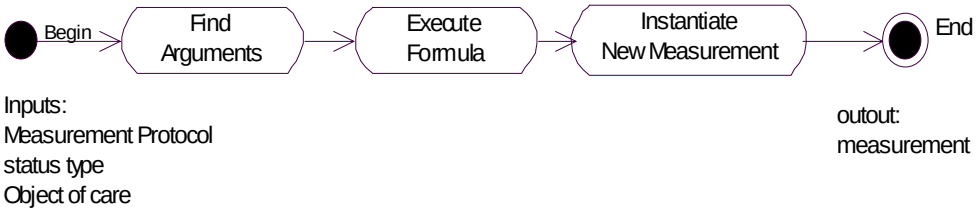


1.2.4 创建测量实例（Create a Mesurement）

从上一节基本了解到了创建 Measurement 对象的大致方法，下面我们来看具体的创建过程，创建过程分为三个步骤：首先找到参数（arguments），然后执行一个公式（formula），然后用所得的结果创建一个新的 Measurement 对象，在这个过程中，参数查找是一个多态的操作，取决于我们要创建因果关系还是比较关系的测量对象。创建过程的序列图如下所示：



活动图如下所示：



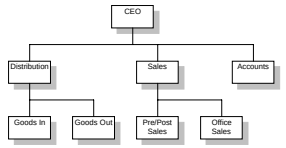
细化因果关系测量的参数查找过程，可以知道，首先要得到所有同一状态类型的测量和测量针对的对象（Object of Care），这些测量目标对象的类型和协议的输入匹配；而比较关系的测量计算公式，则需要查找两个同输入类型匹配的测量实例（同样的针对对象），这两个实例的状态类型分别是比较和被比较的对象。

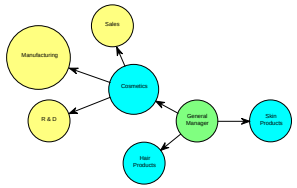
（未完待续）

Organisation

组织模型提供一个可浏览组织的现况及未来的功能，包括组织文化、劳动力结构及能力。

Organisation charts 组织图能说明组织报告的关系及各组织单元之间的关系。





Decision Making Charts 决策制定图是企业高层视各组织单元间的关联，做为下决策时所依循的参考模式。

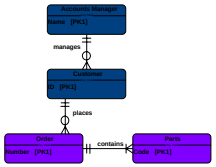
Location

The Location view 是描述商业场所的型态，一般场所及实体的场所。也可能是描述在某建筑物里的场所。

The **Process / Location Type matrix** 显示某个作业程序在那个场所型态里被完成的。

Data Modelling

System Architect 2001 利用实体模型（Entity Modelling）来描述实体、属性及实体间的关系。

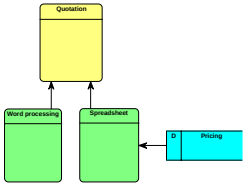


Application

Capture details of the current and future applications software required to support the business functions.

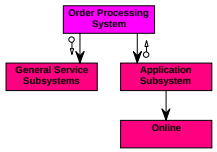
Application Area Map - high-level graphical representation of current applications.

Application Architecture Diagram - 企业高层描述欲建构的应用系统以满足商业的需求。



Application Context Diagram - high-level data flow diagram showing an application with its interfaces to existing or new computer systems.

Subsystem Structure Diagram - decomposition of a subsystem into programs.



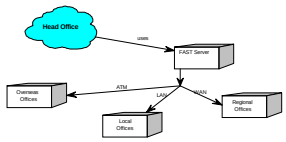
Logical Process Hierarchy - uses an object-oriented approach to form mapping between Business processes and UML.

unified modeling language

Flow Diagram - free-form flow of information or control at subsystem level.

Technology

Network Concept Diagram – 让企业高层了解企业整体的网路概况。



System Architect 2001

Business Process Modelling

Quick Reference

POPKIN SOFTWARE

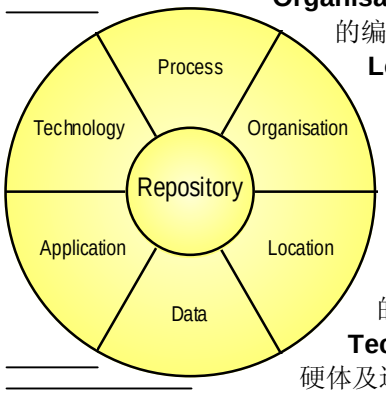
戊丰科技股份有限公司
台北市敦化南路一段 21 号 4 楼之二
Tel：(02)2570-1877　Fax：(02)2570-1876

Methodology Outline

System Architect 2001 支援 企业模型架构（**Enterprise Modelling Framework**） 让您能轻易地驾驭企业的改变及塑造资讯系统模型。

System Architect 2001 支援六个主要的产业元件，任何公司能在完成商业流程分析及设计的过程中都应该考虑到这六个元件。

Process 定义企业要做什么及如何做。



Organisation 定义企业内人员的编制。

Location 定义企业营运的场所。

Data 定义组织内应被记录的商业资料规则。

Application 详细描述软体支援企业结构的功能 。

Technology 说明企业内硬体及通讯设备。

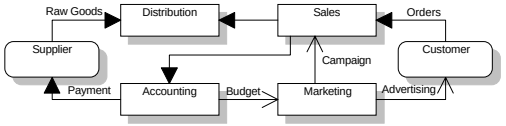
System Architect 2001 Features

- 32 位元使用者界面，及完整的企业模型整合工具。
- 多合一的电脑辅助设计工具（CASE tool ），包含商业描述、物件导向、元件设计及资料模型。
- 支援的法论包括： UML, IDEF0, IDEF3, IDEF1X, Yourdon, Ward-Mellor, Gane & Sarson, SSADM, Catalyst, OMT, Booch and Shlaer/Mellor
- 支援多人使用的网路版本，整合 Microsoft VBA、互动式关联矩阵(Matrix)及影响分析。
- 可以产生 Microsoft Word 文件及 HTML 网页文件。
- C++, Java, Dynasty & Magic – Generation & Reversing, HTML Generation, Simulation.

Business Process

企业流程模型（Business Process Model）提供所有流程及功能的分析，也提供基层作业流程分析，并显示出 动流程及必须的流程，使作业流程能完成工作。

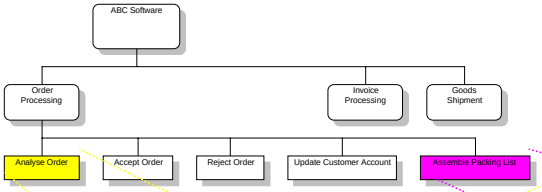
Relationship maps 显示功能间或群组间的相互影响。



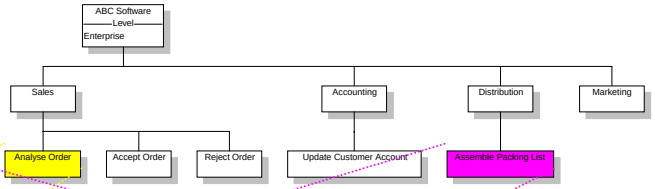
一个 **EBP(Elementary Business Process)** 是商业活动的最小单元：

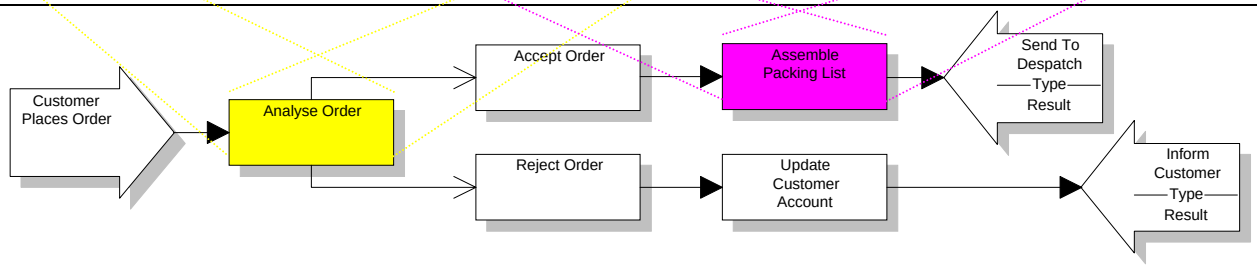
- Is worth examining
- Is performed by or in support of a single user
- May include both manual or automated activities
- Has its user involvement occur at one time

Process Oriented Hierarchy Diagrams 此图是将作业流程依流程群组作分解。

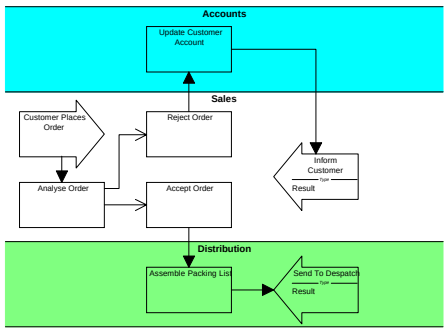


Functional Hierarchy Diagrams 此图是将作业流程依功能群组作分解。



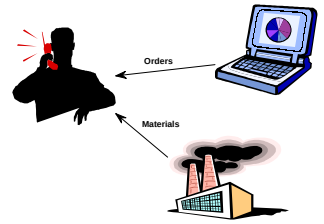


Process Chart 显示一连贯的作业流程。包含： Events, Results, Iterations, Flow Breaks, optionality and exclusivity. **Process Chart** 表示整个或一部份的 Process Thread。

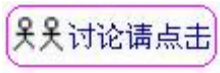


Process Map 规划个别的作业程序应归属于那个组织单位内被完成。

Business Concept 以类似卡通的图案提供企业高层了解作业流程如何工作。



勇于直面需求变更



Windy. J

关键词：需求、需求变更、需求分析、代价估算、面向对象技术、封装、继承、多态、UML、软件设计、软件可维护性、可扩展性、软件可重用性、接口

摘要：作者针对当前软件系统建设中普遍存在的需求变更问题提出了自己的见解，并提出除了从客观上采取加强培训和代价分析等方法外，更重要的是通过采用合理的分析设计方法，进行可扩展性设计可以有效地降低需求变更引起的风险和维护代价，并给出了可扩展性设计的一个具体例子。

软件系统开发过程中的需求变更问题

作为软件开发人员或者软件系统客户，相信我们都遭遇过因为需求变更而需要修改系统的情况，一般说来客户会要求改变界面，改变操作方式，甚至改变业务，说，当时我是那样要求的，不过现在我们的业务调整了…这时需要中断正在进行的工作，需要查证以往的资料，需要修正计划，需要…

需求包括业务需求、用户需求和功能需求。业务需求（Business Requirement）反映了组织机构或客户对系统、产品高层次的目标要求，用户需求（User Requirement）描述了用户使用产品必须完成的任务，功能需求（Functional Requirement）定义了开发人员必须实现的软件功能。在软件系统开发过程中，有很多问题都是由于在需求分析阶段没有正确地收集、编写、协商、修改产品真实需求而产生的，造成这样的状况有几方面的原因：

对需求的理解分歧

当客户向需求分析人员提出需求的时候往往是通过自然语言来表达的，这样的表达对于真实的需求来说是一种描述（甚至只是某个角度的描述），远远不能保证这样的描述可以得到百分之百的正确理解，也

许在同客户交流的第一时刻就埋下了理解分歧的种子，打一个比方说客户说我要的是大象，身子象一堵墙，耳朵象扇子，四条腿象四根柱子，尾巴象绳子，分析人员想，哦，墙、扇子、柱子、绳子这些我都知道，但是真的画出来的时候客户当然会跳起来了！这是理解分歧的问题，一般跟分析员的知识、背景，还有客户表述的标准程度、双方的交流情况有关；

系统实施时间过长

一个大中型系统的建设可能要延续一段时间，当客户提出要求之后，他当时并不能看到系统的运行情况，当双方认为理解大概没有分歧的时候（事实上还会有个 Deadline），开发方就开始工作了。当客户拿到差不多可以试用的产品时他可以实际操作，这时候他就会对系统的界面、操作、功能、性能等有一些切身的体会，有可能提出需求变更要求；

客户具体情况不一

当前客户的情况不一，有可能客户行业的竞争度高，需要随时作出调整和反应，那么他们自然会经常提出需求变更的要求；也有可能客户所在的行业操作不规范，本身存在很多人为因素，这时候开发方更是需要随时准备应变；

开发本身要求

有可能是来自开发方自身版本升级或性能改进、设计修正的要求出现需求变更，这时更是无法绕开这个问题的了！

所以说就算分析人员和客户之间不存在理解分歧，客户对于实际的系统还是会提出一些个人意见，就算没有个人意见，他们自己的业务会变化或环境发生变化，这些都是无法避免的，所以不要梦想那么理想的需求分析，当你开始一个项目的时候就应该意识到，客户需求变更一定会有的，那么对于这样的现状，我们该怎么办呢？客户是上帝，难道我们就象以前一样，跟着客户的需求不停地修改软件，到最后工期延长，员工疲惫，成本成倍增长，客户满意度降低，原来的设计也会改变得支离破碎，系统难以维护？

客观面对需求变更

如果需求一定会变化，如果我们不得不面对，如果我们已经痛定思痛，想要变革，那么还有什么办法可以改善我们的现状？

答案是有的。

加强人员培训

从客观方面可以采取的措施来说，首先，我想不容置疑的是加强对需求分析人员的培训，尽可能增强软件系统、行业的背景知识，提高与客户的沟通能力，增强服务意识和责任感，因为将要开发的系统直接建立在需求分析的基础上；同时规范需求分析人员和客户沟通的方式，以及规范需求说明的格式，如果可能的话，尽量采取象 XP 的 UserStory，或者用户可以理解的用例图来对需求进行标准、规范的描述，保证双方在工具的协助下对需求达到共同的认识，这一点是老生常谈，就不多说。

确定文档的有效性（Validity）

顺便要提的一句是关于文档，需求文档是相当重要的，可是目前存在一种奇怪的现象，本来说必须要有文档，而且是按照某种特定的格式，当然这没有错，但接下来，却没有人关心文档的真正内容是否正确，格式是否真的合理，是否实用（而且很多情况下是在几天时间里赶出来或补上去的），例如我遇到一个例子，需要在原来的需求基础上进行后续开发，文档找到了，完全符合格式的要求，但是我在里面找到的线索是有限的，结果是自己花几天的时间查找数据表结构、甚至查看数据表的内容，询问当时的开发人员，才分析到所要的关系，这种情况在设计文档里也存在，所以同时提一提，希望我们的开发人员、PM 以及各级领导可以注意文档的有效性和有用性问题，甚至对文档的格式进行一下合理性检查。

建立代价估算（Cost Estimate）概念

这一点对开发方和客户同样重要，因为如果出现需求变更，不可避免将带来成本的增加、开发时间延长等不良后果，这样的影响是双方的。

这时候需要区分需求变更的原因，是客户方必要/不必要的要求，还是由于开发方的工作失误，还是双

方都有原因，然后对现实情况进行分析，得出双方实现变更需求的需要的成本，包括时间，人力，资源等等方面，再与客户商讨是否必要进行变更和如何在最小代价下实现变更。

当客户看到实际的代价估算，他们也会再一次慎重地考虑需求变更问题，也会更容易理解系统建设中的进行状况，自然开发方也不用负担所有的需求变更成本，所以进行成本分摊还是有其积极意义的。

当然还有建立需求变更版本控制等等专业的需求管理，在这里不做专门论述。

从软件分析和设计着手

前面说了面对需求变更的几种策略，那么从软件系统分析和设计的角度来看，通过采用合理的分析设计方法，进行可扩展性设计可以有效地降低需求变更引起的风险和维护代价。

采用 OO 技术

采用 OO 技术可以建立易于改变和加强可重用性的软件系统。

对于 OO 技术，我想现在已经不是什么陌生的概念：

- 封装（Encapsulation）可以把问题影响的范围缩小，外部的变化要求对系统的影响可以限定到某个类层次或某些类层次中，从而改变系统的一部分相对简单；
- 继承（Inheritance）可以使改变基于原有技术基础，很大程度上减少重复开发工作；
- 多态（Polymorphism）的应用可以使开发和设计人员在相对统一的接口下更改系统的实现细节，从而改变系统的行为；
- 而且由于对 OO 的类体系结构业界有非常清楚明晰的描述方式，就是目前规范的描述语言-UML，非常易于被开发组的理解并达成共识，促进开发组成员之间的合作以及加强软件开发工作的可延续性；

可见 OO 本身即是一种增强软件可维护性、健壮性以及保持设计稳定性的一种分析和设计方法，本身可

以在一定程度上快速对需求变更进行反应，并可相对减少需求变更需要的成本。（OO 的意义在于分析和设计软件系统的思考方式，以及建立对象库以后的软件重用将给软件系统的开发带来质的改变，但是在建立 OO 开发体系之前的过程，一定会是一段荆棘遍布的路，需要付出加倍的努力以及达成思想的转变。这里还有一个误区需要澄清的是很多人以为用了 C++，PB，VB，DELPHI 就是面向对象的开发了，其实只是用了一些面向对象的工具，骨子里仍然是结构化的分析和设计方法，套上一层 OOP 的外壳而已。）

可扩展性设计（Extensible-Design）

其次，从我们可以控制的软件设计来说，怎样进行合适的设计才能最大程度减少需求变更带来的代价？

也许有人说，我的设计极为灵活，我已经预计了客户可能提出的要求，并设计几种应对的方式，到时候客户提出来，呵呵，我已经解决了。这样的想法不错，至少比僵硬的设计强，但是谁可以保证设计者可以预知以后的需求变化？而同时为了达到这种灵活（万能/多能？）的设计，设计将变得复杂，而且可能那些多余的设计从来不会被用到？复杂的设计将增加实现的难度和提高成本，并有可能带来潜在的 Bug，使得系统难以维护。

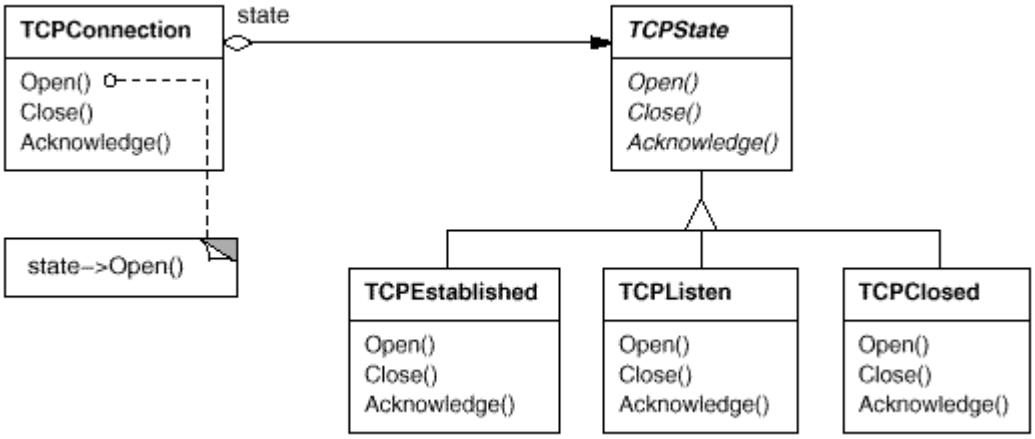
设计的思想应该有一些小小的转变，那就是，设计确实要灵活，但是要体现在可扩展性上面，也就是说，设计可以简单，但是一定要易于转变，需要给出便于改变的接口，这一点很重要。

例如，现在有一个类叫做 TCPConnection，来代表计算机网络通信中典型的 TCP 连接，对于这个连接而言，它可能处于以下几种状态：Established（连接已建立），Listening（正在侦听），Closed（连接关闭）。一个连接对象需要从其他的对象接受请求，至于它的反应则决定于连接对象所处的状态，对于（打开连接的请求），如果是在连接关闭状态，则进行 Open（），处于其他状态则不做反应；同样，如果在连接建立和侦听状态，可以进行 Close（），在连接建立状态可以进行 Acknowledge（），即接收数据。

对于这样的状况，最不可取的设计应该是用一系列的 Switch 语句（甚至 If/else 语句）进行 Hard 设计，对于以后每一次需求改变，都需要改变源代码，接踵而来的系统一致性、文档更新等工作将使开发人员不可避免地陷入一场灾难，这样的后果将导致原来就不合理的设计变得更加支离破碎，系统维护的代价将越来越大；就算没有需求变更发生，这些设计的可重用性也会极差。

稍好一些的设计是预先估计并设置 TCPConnection 类所有可能的状态，并预先加入设计，这种需要付出更多的设计、开发、维护的代价，而且也很难达到完美的效果，所以不多说了。

下面介绍一种经典的设计思路，这种设计可以充分体现“为（系统）将来改变预留接口”的可扩展性（Extensible-Design）思想，并且很好的实现了这一思想。在这里，我们引入一个抽象类 TCPState 来代表 TCPConnection 类的状态，给出具体各种状态的通用操作接口，并派生出不同的子类（实现具体的操作）去实现 TCPConnection 类的不同状态，例如派生出 TCPEstablished 类来实现 TCPConnection 类的连接建立状态。结构图示如下：



只需要在 TCPConnection 类中包含一个 TCPState 的状态引用，并在 TCPConnection 的状态改变时更新为当前的状态引用，例如在连接关闭时进行 Open（），状态引用就应该从 TCPClosed 变成 TCPEstablished，这样就实现了原来的要求。

但这个设计思路的意义远不止于此。我们可以看到，抽象类 TCPState 已经为 TCPConnection 类将来可能的状态留出接口，只需要不断派生具体的不同状态子类就可以实现将来的状态变更，并且无须影响原有的设计，也无须加入多余的代码来实现现在还不需要的功能，所以这是一个优美的、可扩展的设计思路，非常清晰，易于维护，相信可以给我们在做软件设计时带来一些启发。

结论

可见，在面对需求变更时，除了客观上可以通过人员培训、代价分析等管理方式进行有效的需求管理

外，从分析和设计的角度可以通过采用合理的分析和设计方法，还有改变我们设计的意识，可以做到对需求变更的灵活应对，至少可以在一定程度上降低维护代价和提高用户满意度。

软件需求的管理和控制是非常专业的学问，作者在这里结合自己的实践提出一些粗浅的认识，只是想起到一个抛砖引玉的作用，希望大家可以一起来面对和想办法解决我们在系统开发过程中的实际问题，我想那样才是我真正想达到的目的。

参考文献：

1.《软件需求》，（美）Karl E. Wiegers 著，陆丽娜、王忠民、王志敏等译，机械工业出版社，2000

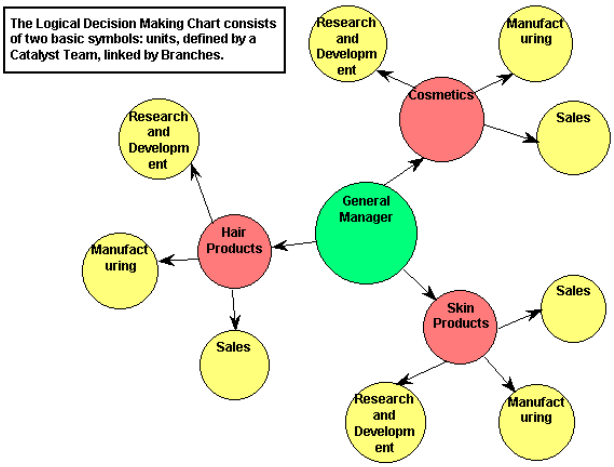
2.《Design Patterns:Elements of Reusable Object-Oriented Software》，（美）Erich Gamma、Richard Helm、Ralph Johnson、John Vlissides 著，Addison-Wesley，1994

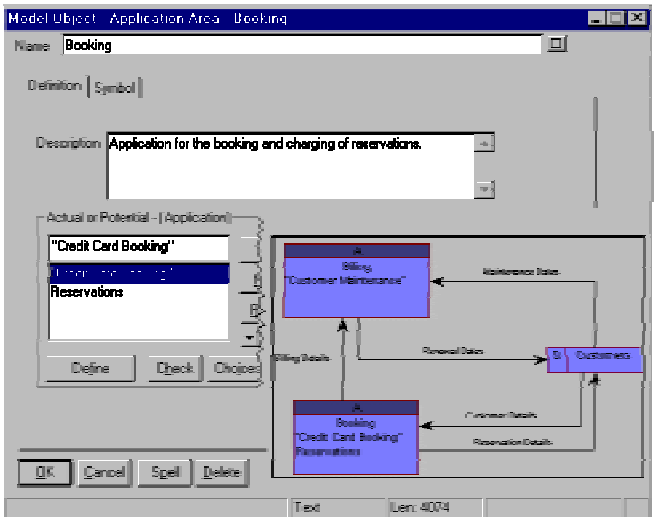
3.《Is Design Dead》，（美）Martin Fowler，Thought Works，2000

Face the Requirement Changes

Abstract: This article illustrates the general requirement-changes during software system construction. It discovers many useful ways such as Object-Oriented technology and Extensible-Design to resolve the problems mentioned above, other than the enhancement of training and cost analysis. Also an Extensible-Design example is involved in this article for better understanding.

Keywords: Requirement, Requirement Changes, Requirement Analysis,Cost Estimate,Object-Oriented technology, Encapsulation, Inheritance, Polymorphism, UML, Software Design, Software Maintainability, Software Extensibility, Software reusablilty, Interface





System Architect® 2001

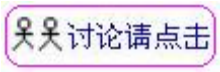
更多关注业务而不是技术的电子商务解决方案

POPKIN SOFTWARE 的旗舰产品，融业务流程建模、UML 设计、关系数据库建模和结构化分析于一体，被许多“Fortune 500 强”公司应用，誉为 "理想的全程企业电子商务解决方案"。

http://www.popkin.com

对此产品发表评论>>

大中华区发行：Magic Consultant Enterprise
UMLCHINA 提供技术支持



掌握可用性规则

Lucy Lockwood 著，Nothing 译

细致的用户界面设计是高质量软件的标志之一。设计良好的用户界面使用户与软件的交互更有效率、更能减少用户的错误倾向，更快地学习和更富有成果。好的界面设计不仅仅是屏幕布局周全的考虑，它着眼于在充分理解需完成的工作的基础上创建一个用户界面与系统的有效架构。下面这些规则将能帮助开发小组提高项目界面设计的质量。

1. 为实际工作设计

软件是一个使能工具，让我们更快更有效率地处理事务并延展我们的能力。除了一些入魔的技术专家和全心的质量至上者外，大多数人使用软件不单单是为了享受运行其他人代码的乐趣。不管是用它在互联网上查询明天的天气还是控制微波炉的热量来加热汤，或是为病人远程诊断，软件总是一个工具，一个实现目标的方法。考虑到这点，开发者和设计者就能正确地理解用户要完成的工作和为了成功实现用户需求软件应提供什么。

关注实际工作需要考虑为什么用户在完成一项任务时要采取不同的操作，注意不要迷失在实际工作之外的管理和其他事务上。同时要避免技术的诱惑。实现一些很酷的软件特性而试图解决实际不存在的问题，结果只是使用户界面混乱和易于混淆。

留出时间和努力去充分收集相关信息和分析软件用户的需求。让用户参与到这个过程中，但不要盲目接受用户或客户提出的功能需求。毕竟，用户可能是应用领域的专家，但他们对软件的设计和分析有可能并不熟悉。保证在开发过程中引入一种有效的方法，能够收集、组织、验证支撑工作的信息。学习问题领域的描述语言，掌握工作的流程，理解每个需求是怎样以及为什么有机结合成为一个整体项目。然后整合分析成果到界面设计过程中并应用这些认识驱动整个软件的设计。

2. 不要急于具体化

开发者一贯倾向成为一个解决方案的提供者，喜欢解决问题并希望能尽快看到努力的结果。有时这种快速获得结果的倾向性导致的是一个平庸的解决方案。现今的可视化开发工具更是鼓励了这种倾向，纵容我们简单地通过在屏幕上拖放预先定义的组件来解决界面设计问题。举例来说，通常很少考虑组合框还是一个下拉列表是最好的选择。在开发早期不要急于具体化，开发者和小组在开发过程中可能创建更可用的设计并提出高超的用户界面设计方案。不要过早定义实现的细节，而应在更好理解所需完成的工作和更好把握用户工作流程中的每一个步骤的意图之后。如果开发小组已经跳跃到实施具体方案的阶段，应注意在设计过程中将那些想法置于一个“反馈—提出”的循环中。

一些技术如抽象原型，能以一种通用的方式在贴纸上表达必需的用户界面元素，可以帮助开发小组先关注整个用户界面结构。抽象原型技术能够保证软件提供所有必需的组件，也能保证逻辑地安排这些组件，而非早早地进行界面图形设计和界面组件选择。

3. 避免为创新而创新，不要成为时尚的奴隶

界面设计中新的东西并总不是好于旧的东西。应用得当的话，新的界面组件和交互设计方法将是一个强大的工具并将占领市场优势。过不多久界面设计领域里真正先进的的方法就会被拷贝流传而成为标准设计词汇的一部分。但是，许多创新的界面设计比传统的界面组件和交互设计可用性更差。

另外要注意的是避免为艺术而创建艺术品。我所见到的一些可用性最差的软件就是图象艺术家或即将成为图象艺术家的作品，他们迷失在视觉效果里最终导致不可辩识的控件、可读性差的布局、不能工作的导航设计。效率和美学并不总是互斥的，相反，最好的软件是两者的结合。虽然人们的艺术的才能总是有限，但是可以集中注意力创建有效、高可用性的满足用户需求的软件。

最后，不要假定界面设计应模仿一种特别的微软模式或应用至少一种最近的界面组件。用户界面设计有自己的流行趋势，重要的是保持现状。新的组件和交互术语（idiom,Alan Cooper 的称呼）每天都在产生。就象房子、车子或者衣服，软件的风格在变化和进步，但只要简单地看看用户界面设计就可以跟上当前系统设计。特别地，如果设计的软件的运用与微软的应用紧密结合或关联，在很多情况下，这是一个合理的选择。然而，追随领导不是一个通用的解决方案。有一个例子，是我领导评估一个模仿 Microsoft Excel 界面设计的货币交易系统的可用性。设计的决策使得用户可视分析信息非常困难，从而导致笨拙的、低效率的导航。货币交易员需要的新交易系统并不是微软电子数据表的界面设计。

4. 努力建立有效的交互

没有人喜欢乏味、笨拙的工作。确认你的软件没有增加其他人工作的不愉悦感。计算完成一项普通工作所需的击键次数和点击鼠标的次数。让简单的事情简单处理、常用的交互快捷完成。不要将重要信息或关键处理淹没在多个窗口之下或者隐藏在一个“名不符实”的菜单里。应用建模技术如导航映射来帮助设计高效的工作流和避免过长或导致死结的导航路径。只要在纸上画一个原型，用户界面设计的原则如基本效率、任务一致性和任务可见性就可以用上，并且能指出问题所在区域。

正如令人难忘的 Dilbert 的漫画指出的一样，一些软件开发者为用户设计了额外的无意义的步骤，惩罚用户犯下的任何错误，让用户象跳圈一样完成日常工作。如果小组里某些成员显得憎恶用户，让他去见见临床医生，而不要在“Edit”菜单里加上”Other”。

5. 为实际工作试用界面

作为最后一个保证用户界面设计质量的措施是：让每个设计和开发软件的成员最少连续半小时使用它，连续使用一个小时更好。使用实际的数据——用户实际使用的数据。如果由于保密或安全的原因不能使用实际工作的数据，可以让客户或用户建立足够多的“伪数据”（不仅仅是 10 到 20 条记录），能精确反映实际数据的复杂性、潜在的错误和模糊性（subtleties）。作为一个独立的校验，确信测试了所有数据域的最大长度，正确性和过率检查。经常令人惊讶的是在数据库正确定义的域在用户界面上却是长度不够或定义不当。如果定义的数据域是 25 个字母的长度，试着输入 25 个字母。你能在屏幕上见到所有的 25 个字母吗？如果需要的话，邮政号码的输入框能处理国外字母和数字混合的编码吗？

是的，在许多公司这些问题都会归结于质量保证和数据库设计部门，但是，令人吃惊的是它一直被疏忽。所以，应用用户界面试用的经验来测试系统自身。注意到测试的过程象是汽车装配线的最终检查——客户将要看到和体验到。如果界面包括了数据输入域，输入 50 或 100 条记录，而不仅仅是 1 到 2 条记录。如果应用程序工作在不定的光照和噪音情况下，在这种环境条件下试用它。如果必需快速输入或屏幕快速反应，将开发小组所有成员置于同一个时间限制的压力下，然后问自己：每天我都想使用这个软件吗？我希望这些技巧能帮助你或开发小组回答：“是的”。

原文链接 http://www.umlchina.com/GUI/learning.htm

KCOM 技术介绍

KCOM 是 KCOM 公司独立开发的一系列开发平台和计算机语言技术的总称。
它包括：KCOM 组件模型，KCOM Basic 语言，KCOM Stage 运行环境，KCOM Space 开发平台四项技术。

KCOM 组件标准：
KCOM 组件标准是一个完善的组件标准，可以自定义属性，方法和事件，并使用事件驱动的方式编程。同时源代码方式存储有利于在网络上传输。

KCOM BASIC：
语法结构完善，易学，可以使用中文进行编程，例如使用中文变量名称，中文方法名称，属性名称等等。独有组件运算功能能够实现代码的可视化。

KCOM Stage：
KCOM Stage 是 KCOM 组件的解释器，由于采用了代码的结构化存储，在解释之前不需要对语句进行扫描和文法处理，同时采用了高效的寻址方式，所以比一般的脚本语言的解释效率要高。

KCOM Space：
能够方便的编写 KCOM 组件和代码，是一个可视化开发平台，具有丰富的界面排版，代码编写和代码调试功能。

KCOM Space 以其自定义的组件标准(KCOM 组件标准)为核心，完整地实现了一门计算机语言（KCOM Basic)、一个完整的组件化开发平台（KCOM Space）和一个高效的虚拟机（KCOM Stage)。作为国产软件在高端领域的突破，KCOM 表现出了与世界同步的技术水准。由于采用了全面组件化的思想，抛弃了传统的设计方法，全面革新了开发平台从底层的语言到上层的操作界面的设计。

KCOM 公司的产品在 2000 年中国国际软件博览会上获得创新奖。并受到倪光南院士的高度评价。

KCOM 公司合作意向：

KCOM 公司在长期对于开发平台和计算机语言技术的关注与不断开发的基础上，形成了自己独特的技术特长与优势。在如今的软件领域，软件和开发平台日益紧密成为趋势，其中突出地表现在：

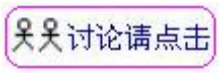
软件与基于该软件的二次开发平台：如 MS Office，Notes，GIS 应用软件及开发平台，3D Max，AutoCAD，Director 等等应用软件。 一些稍小的软件如 InstallShield 也在其中使用了自己定义的解释语言，以提供强大的扩展功能。

企业流程快速建模与软件生成工具：此类工具在国外的企业应用解决方案市场中被大量使用。而国内的系统集成企业正在开始开发和形成自己的解决方案与定制工具。

嵌入式企业应用前端设备：该设备是我们所看到的一个巨大的市场机会，基本原理在于软硬件一体的显示与操作前端，采用 NC 体系，运行企业应用。

在以上几个领域中，KCOM 公司都能够以自己长期的积累和技术特长，与合作方展开广泛的技术合作，希望能够来信探讨：zhangiii@163.net

http://www.kcomspace.com.cn/



在小型软件开发组织中使用 CMM

Mark C. Paulk 著，张俊 译

CMM SM 是适用于小工程项目和小规模组织的经剪裁的 CMM 版本。而 CMM V1.1 是用适宜于那些和政府签约的大型组织的标准实践来表达的，这些实践必须剪裁才能适合不同于这个样板的组织的需要。

摘要

由美国软件工程学会（SEI）开发的软件能力成熟度模型（CMM, Capability Maturity Model），已经成为软件过程及质量改进方面的世界主流。尽管 CMM 被广泛接受了,但有关如何在商业驱动的软件过程改进中有效地使用它，特别是针对小型组织和小型工程项目，仍存在着许多的误解。有关在小工程或小组织中应用 CMM 的一些常见问题包括：

- 什么样的才算做“小”？标准是根据人？时间？项目大小？还是产品的艰难复杂程度？
- 什么是 CMM 的“需求”？是否有不该被应用到小项目/小组织中去的关键过程区域或目标？有好的过程“ 不变量”吗？
- 造成 CMM 被滥用的驱动力和动机是什么？

这篇论文以小型组织为例讨论了怎样在各种商业环境中正确而有效地使用 CMM。从那些对改进其软件过程感兴趣的任何组织得出的结论是：使用为大组织/大项目所开发的发行版来相应地解释用于小项目或小组织中的 CMM，可能会存在程度上的差异，但它们决不会是本质上的不同。正确有效地使用 CMM，要求具有专业性的判断，并且能够理解 CMM 是如何针对不同的用途来进行建构的。

作者简介

Mark 是美国软件工程学会（SEI）的一名技术人员。他 1987 年加入 SEI ,最初参与的是软件能力评价项目的工作。Mark 从一开始就工作在软件能力成熟度模型方面，并且是开发 CMM1.1 版本时的项目领导人。他也一直活跃在制定有关软件工程的标准上，这些标准包括：

- ISO 15504 (也称为 SPICE--软件过程改进和能力决断),一整套软件过程评估的国际标准
- ISO 12207 , 软件生存周期过程
- ISO 15288 , 系统生存周期过程

在加入 SEI 之前，Mark 是系统开发公司（System Development Corporation，即优利国防系统公司 Unisys Defense Systems 的前身）的位于亚拉巴马州 Huntsville 的弹道导弹防护高级研究中心（Ballistic Missile Defense Advanced Research Center）的一名高级系统分析员。

Mark 在范德比尔特大学获得了他的计算机科学硕士学位。在 Huntsville 的亚拉巴马州立大学获得了他的数学和计算机科学学士学位。

专业资格及证明

- 电子学会高级研究员及电子工程师（IEEE，美国电气及电子工程师学会）
- 美国质量协会高级研究员（ASQ，美国质量协会)
- 美国质量协会（ASQ）认证软件质量工程师

※※※

1.　介绍

软件工程学会(SEI)是由美国国防部 1984 年设立的一个联邦资助研发中心，主要通过大范围承包工程的研究来寻求软件工程技术的跃迁——即改进软件工程的实践。在某种意义上, SEI 的存在是“软件危机”（即习惯性的拖延，超出预算，达不到预期功能，以及不可靠的质量等[Gibbs94]）的结果。硬性地讲，危机大多是由软件自身造成的，一位首席信息官曾说过：“我宁愿让它有了错，也不想让它出得晚，稍后我们总会去修理它的。”许多组织要完成预期成本和进度目标的关键就在于——经常性地关注质量成本，一再地学习以往 20 年由美国工业界所推演出的现称为“全面质量管理（Total Quality Management，TQM）”的一门课程。

引用 DeMarco [DeMarco95], 各种因素综合造成的如下情形并不令人惊奇：

- “人们总是抱怨我们，因为他们知道在抱怨的时候，我们会努力工作。”
- “大多数软件评估报告是令人沮丧的……但好在他们还不是对评估过程完全不满。”
- “正确的进度表是绝对不能达成的，只不过还不那么明显地不能达成。”

DeMarco 继续观察到我们的工业界正在疲于奔命,而感觉上唯一真切的选择就是降低质量以换取速度。

TQM 课程聚焦于在质量管理引导下减少开发周期，提高生产率，增进客户满意程度并努力取得商业上的成功。挑战，当然被定义成“聚焦质量”的实用手段是什么，随即就要系统地寻求各种质量问题。也许 SEI 最成功的产品就是软件能力成熟度模型(CMM)，就是说，它给在全世界软件社团中已成为主流的软件过程改进提供了路标[Paulk95]。CMM 定义了怎样使开发组织的软件过程走向成熟的 5 个等级的框架结构。这些等级描述了从特别紊乱的混沌过程到成熟的、有纪律的软件过程的进化路径。如图 1 中的概示，5 个等级和 18 个关键过程区域详细描述了它们。5 个成熟度等级指明了对于成功的过程改进（即在许多案例的研究和调查中被文档化记录的有效性）的优先顺序[Herbsleb97,Lawlis95,Clark97]。

成熟度等级 **Level**	**焦点** **Focus**	**关键过程区域** **Key Process Areas**
5 **优化级** **Optimizing**	*持续不断的过程改进* *Continual process* *Improvement*	缺陷预防 Defect Prevention 技术变更管理 Technology Change Management 过程变更管理 Process Change Management
4 **已管理级** **Managed**	*产品和过程质量* *Product and process* *Quality*	定量过程管理 Quantitative Process Management 软件质量管理 Software Quality Management
3 **已定义级** **Defined**	*工程化过程及组织支持* *Engineering processes and* *organizational support*	组织过程焦点 Organization Process Focus 组织过程定义 Organization Process Definition 培训大纲 Training Program 集成软件管理

		Integrated Software Management 软件产品工程 Software Product Engineering 组间协调 Intergroup Coordination 同行评审 Peer Reviews
2 **可重复级** **Repeatable**	*项目管理过程* *Project management processes*	需求管理 Requirements Management 软件项目计划 Software Project Planning 软件项目跟踪和监督 Software Project Tracking & Oversight 软件转包合同管理 Software Subcontract Management 软件质量保证 Software Quality Assurance 软件配置管理 Software Configuration Management
1 **初始级** **Initial**	*能人高手和英雄行为* *Competent people and heroics*	

图 1. CMM 概况

尽管 CMM 的当前发行版本 1.1 的焦点是在与政府签约的大型组织和大型工程项目上,但 CMM 是被设计成了为实现对软件工程及项目管理的详细指导及示例而普遍适用并分级细化的抽象模型。CMM 的关键过程区域，是以能达成被描述为关键实践、子实践及示例的目标而满足的。CMM 用以评估的构件是成熟度等级、关键过程区域和目标。另外的构件则在怎样解释模型上提供了信息和指导。18 个关键过程区域共有 52 个目标和 316 个关键实践。虽然 CMM 的“需求”可以被概括成对 52 个目标的描述，但它的支持材

料却包括了将近 500 页的信息。关键实践和示例描述了好的工程和管理实践是什么样，但是没有说明如何去实现这些过程。

CMM 之所以能成为一个指导过程改进的有用工具，是因为它历史性地成为了通过对软件社团已开发软件的广泛评审而形成的全面质量管理(TQM)概念的一个常规应用。它的五个等级是非常简单的，但如能将其巧妙运用，就会找到一个激励人的支点，正如美国国防部的程序经理所坦陈的：“底线是进度表。我的晋升和加薪，就全靠首当其冲地完成进度表了。”

正当 CMM 由于对指导软件过程改进的重要性而令人鼓舞地在全球变得流行起来的时候，它也正被一些人滥用和误用，并且另外也有人没能有效地使用它。CMM v1.1 提供的指导往往是面向大工程和大组织的，在使用过程中小组织发觉到这方面有问题，尽管我们坚信 CMM 的基本概念对于任何组织规模、任何商业环境、任何应用领域都是适用的。

难道按时完成进度、预算和需求，对于小项目或小组织来讲，真的那么重要吗？在某些环境下，这确实是值得商榷的，就如同商用小塑胶袋，拿它跟那些市场份额总是占先的装船出口的顶好产品相比，其成本是极其微不足道的。如果一个组织的雇员十分满足安于现状，那么 CMM 所提供的能导致真实变化的东西将会很少；变化只会出现在那些对现状非常不满并且乐于标新求异的经理和员工身上。无论对大组织还是小组织而言，这都是无庸置疑的。

CMM 在称心如意地实施管理及工程实践方面，都提供了良好的建议，它强调以人为核心的管理、沟通和协调以及能体现软件开发和维护特性的强化设计的过程。无论如何，把它看成灵活的指南而非生硬的指令吧，还有对于软件工程和管理，以及应用领域和组织的商业环境等方面，CMM 用户必须能够在这些方面应用其基于知识和经验的专业判断。因为 CMM 聚焦于软件，所以 TQM 的一些重要方面不能直接照搬到模型里，比如系统工程里的“人的问题（people issues）”和“较宽泛的审视（broader perspective）”，这些在商业上可能是至关重要的。CMM 应该被看成使用在软件过程改进系统步骤里的一种上下文工具，比如 SEI 的 IDEAL 模型，如图 2 所示[McFeeley96]。

在对软件过程改进的讨论中，开始的问题总应该是这样的：为什么软件组织会对使用 CMM 感兴趣？如其愿望是以直接依从商业目标以及心甘情愿投身改革而来改进过程，那么 CMM 确实是效用非凡、功能强大的工具；如果 CMM 只被当成单纯的短期时髦，那可真是把一剂糟糕的药方拿到了手。如果驱动力是客户利益，理想情况下客户利益将导致客户和供应商之间协作的改进。有时供应商的利益集中在软件能力评价(Software Capability Evaluations，SCEs)上，如此则可在那些来源选择及签约监督方面是由政府获取代理的项目上有所表现。国防部在执行软件能力评价(SCEs)标准的政策上是排斥绝大多数小组织和小项目的

[Barbour96]，但也存在它们可以有所作为的机遇。

很多 CMM 的滥用都对“其他人”可以做什么的担心置于不顾。如若一个开发组织能在用 CMM 来导引胜过被需求来牵引上达成共识，那么在模型中要解决问题的很大一部分就化解掉了。有很多这样的案例。尽管如此，那些对于好的工程和管理实践的无知仍将成为问题。对于那些只有很少的管理方面的经验和训练而只是因技术优秀就被提拔到管理层职位的人来讲，问题是显而易见的。由 DOD（美国国防部）特种部队确认并提出的问题是[DOD87]：

- “少数区域在最佳当前实践和平均当前实践之间有着如此巨大的缺口。”
- “大问题不在技术方面……现今军事软件开发的主要问题不再是技术问题，而是管理问题。”

2. 小组织和小项目

本篇论文的焦点对准了在小组织正确而有效地使用 CMM，因为经常有人问我，“CMM 是否能被用在小项目(或小组织)？”然而有关“小”的定义却是模糊难解的，如表 1 所示。 在某段时间里我们致力于为小项目和小组织而开发出剪裁适当的 CMM，1995 年 CMM 剪裁工作室的结论是我们甚至不能对什么才算“小”的真正含义达成一致意见！这个结果得出了一篇更注重于“如何剪裁 CMM”而不是“已为小组织而裁好的 CMM”的报告[Ginsberg95]。1998 年 SEPG 会议关注于 CMM 及小组织上，“小”被定义成“5 个或更少的人为期 3 至 4 个月的开发”。Brodman 和 Johnson 则定义“小组织”为少于 50 个软件开发人员并且“小项目”为少于 20 个开发人员[Johnson98]。

表 1. 定义一个小项目

小的变体	人数	总的时间
小	3-5	6 个月
很小	2-3	4 个月
微小	1-2	2 个月
个人	1	1 星期
荒唐！	1	1 小时

注意从小到微小的项目是在被 Humphrey 称为小组软件过程（Team Software Process，TSP）的范围里，而个人的开发努力则在个人软件过程（Personal Software Process，PSP）的范围里[Humphrey95]。TSP 和

PSP 阐明了 CMM 的概念是如何应用到小项目中的。“荒唐”变体则描述了一个解释性的问题。在两种场合里，变体都会被论述，问题是“项目”的定义。两种情况下它都是个维护环境，而且组织的“项目”将被描述为 CMM 的任务；更多关于 CMM“项目”的精确解释是一个基线的升级或者维护的发行……但术语的冲突会将其搞乱。

对于那些使用 CMM 的小组织来说，首要的挑战就是其主要商业目标要能存活下来！甚至在决定之后，现状是不能令人满意的而且过程改进也将有助于发现资源并为过程改进分派职责,接下来通过制定与部署过程所要做的是一项艰难的商务决定。小组织往往相信：

- 我们全都是胜任的——人们是被雇佣来做工作的，我们可不想负担那些要在雇佣期间进行培训所花的任何时间或金钱
- 我们全都彼此沟通——因我们是如此紧密地在“渗透式”工作
- 我们全都是英雄好汉——我们无论做任何需要做的事情，规则都不适用于我们（这些恰恰达成了将工作完成的方式），我们承受住了短周期及高压力

然而对于小组织，也正象大组织一样，有着非文档化的需求所带来的麻烦，以及无经验的经理人员、资源分配、培训、同行评审和产品文档等方面带来的问题。尽管有着这些挑战，小组织仍能非同寻常地进行创新和提高生产率。尽管有一大把需要人们去解决的大块问题，通常小组织比大组织更具生产效率，他们能更敏锐地成形生产要素并且远远更少见有沟通方面的问题。无论如何，遗留下来的问题是，小组织也需要过程纪律吗？回答这些 CMM 咒语,我们需要考虑到纪律包括了什么？而那将引入这篇有关 CMM 指导性课题论文的核心。

即便如此，评估小组织时运用最新式的评估过程是明智可取的。为期两周的 CMM 内部过程改进基础评估(CMM IPI)的形式也许是多余的[Strigel95, Paquin98, Williams98]，甚至会由于缺乏监控而导致某些遗漏，而关键是有效地确认重大问题。我建议将注意力集中在建立在企业文化上的制度化实践方面：如规划、培训等等，并确切地将过程改进落实到商业需要上。

3. 解释 CMM

CMM 都适用在什么地方呢？CMM 是按照在任何环境中为任何项目都能提供良好的软件工程和管理实践来塑造的。模型是被分层次描述的：

成熟度等级 (5)

->关键过程区域 (18)

->目标 （52）

->关键实践 （316）

->子实践和例子 (许多)

根据我最近十年来在软件过程工作方面的经验，阐述的环境以及 CMM 的剪裁需要包括：

- 非常大的程序
- 虚拟项目或组织
- 地点上分布式的项目
- 快速原型法的项目
- 研究及开发组织
- 软件服务组织
- 小项目和小组织

对于小项目和小组织的解释性指导也同样适用于大项目和大组织。在正确有效地使用CMM的过程中，智慧和常识是必要的[Paulk96]。所有（软件）项目都有所不同，但所有（软件）项目也都有所相同——这可全都是真的。我们必须平衡现实：相似与唯一，秩序与混乱。那些成功所缔造出的持久组织[Collins94]是真正有能力的学习型组织[Senge90]；此外在其他地方也必须得益于其成功。

CMM 的标准构件是成熟度等级、关键过程区域和目标。CMM 的所有实践都是有益处的。既然那些详细的实践都首先支持大型签约的软件组织，也就没必要正好写成是直接适用于小项目和小组织——然而它们同样提供了如何达到目标和可重复地实现已定义的、规范的、不断改进的软件过程。这样就会避免了过程评估方式上的简单武断——诸如“去问老张吧”。

我最通常的指导性建议是开发出一套适用于组织的置于 CMM 术语和语言间的映射。特别地,组织结构的期限行为、角色、关系以及过程的形式都需要映射到其组织的相应部分，以防止出现无法理喻的事情，诸如“荒诞的一小时项目”。组织结构的例子包括诸如质量保证、测试及配置管理等“独立小组”。应该用术语明确地指定适当的组织角色，诸如项目经理及项目软件经理等。人们可以担当多重角色，例如，一个

人可以同时做项目经理、项目软件经理、软件配置管理（SCM）经理等等。明确规划好这些，将生发出对 CMM 更生动简洁也更协调一致的阐明。

一旦术语问题被理解了，我们就得考虑什么是守纪律过程的“不变量”（invariant，一个必须永远为真的约束）以及哪些实践依赖于上下文关系。除了软件转包合同管理（即若无转包合同的话就可能成为“不可用的”），一般来说我们总是假定关键过程区域及目标关联到任何环境。我想象不出同行评审被等级 3 的组织适当剪裁掉的情形。尽管所选择出的实践（诸如正式方法）可以替代同行评审，但这还是一个需要足够专业性判断的问题。拥有专业性的判断和训练、经验丰富的评估人员是至关重要的，甚至对小组织也一样！[Abbott97]

我从没见过下列哪个环节是不必要的(尽管实现方式不同)：

- 将客户(系统)的需求记录成文档
- 与客户(并且最终用户)沟通
- 同意承诺并签约
- 计划
- 文档化过程
- 工作细化结构

当然，一些实践都被处理成了“大项目的实现”。小的项目未必需要软件配置管理组或者变更控制部门……，然而配置管理及变更控制总还是要有的。一个独立的软件质量保证(SQA)组也许不是必要的，但目标的认可始终是需求要被满足。一个独立的测试组也许没必要设立，但测试总是必要的。即使小组织和大组织之间的实现有着根本的不同，但我们看到对于上下文相关的实践，其意旨是建设性的。如果有人读到 CMM 所定义的“组”，它是被陈述为“一个组可以是被分派兼职的单独个人、从不同部门分派了几个兼职的个人，也可以是全职的几个个人”，这里就有意迎合了环境的变化。

除了这些，一些特殊的问题再三地出现，尤其是对于小组织，涉及有：

- 管理资助
- 度量
- 软件工程过程组(SEPGs，Software Engineering Process Groups)

- “不保修”过程
- 文档化过程
- 剪裁
- 培训
- 风险管理
- 计划
- 同行评审

获取高层管理的赞助是构建组织能力的关键成分，这看起来固然很陈腐老套。做为个体，我们可以在我们可操控的范围内磨练自身的专业素养及自制能力。可是若要一个组织作为一个整体来改善其效能，那么其高层管理就必须积极地支持变革。由下至上地改革，无须赞助和协同，却能够导向超过可预期改进的组织能力的胜地，这可真是奇闻了。即便如此，当总裁（或者公司创始人）是主角时，敬重“冠军”往往是具有撼动整个组织的影响力的——包括总裁本人。

对于大多数组织而言，管理实际上是必须基于一个度量基础的范例转换。掌握有用的数据，就是说你必须了解性能数据的含义以及如何有价值地去分析它。从收集简单的有用数据开始，你也不得不敏感于经由度量而导致紊乱行为的潜在因素[Austin96]。度量活动要确认什么是重要的，但一些事情是难于度量的。管理需要确保注意力倾注在项目的所有可评价方面，包括那些难于度量的，而不仅仅是那些易于度量和跟踪的。

在大多数组织中，软件工程过程组(SEPG)或一些类似小组应该成为协调过程定义、改进及部署活动的工作组。把资源奉献给 SEPG 的原因之一是要确保完成评估调查。许多改进纲要仅仅因为有着不是从评估所导致的行动而失败。小组织可能没有全职的 SEPG 人员，但改进的职责应该明确地加以指派和监控。

起始于“不保修”（"as is"）过程，而不是“合理”（"should be"）过程——对于有效实施的实践与对指派任务的抵触来说，这相当于两者之间的杠杆，此起彼落。要求自上而下的每个人都遵循的“合理”过程，若其显著地不是由被授权的工人所参与开发的，则将是一个通用的失败处方。“不保修”过程进化的理由是人们从事工作是希望工作任务能被完成（即使那意味着要整天围着系统转）。“合理”过程或可行，或不可行——只有在特定的文化环境中才能成为可行。伴随着过程管理和改进的组织焦点，“不保修”和“合理”过程将聚合在一点——即导致有组织地进行学习。

文档化你的过程。文档化地记录一个过程(或产品)的原因是 1)沟通——给别人一个当前的交待以及给自己一个今后的备忘；2)理解——如果你不能写下它，你就不能真正理解它；以及 3)鼓励连贯一致性——拥有可重复的优势。文档化过程支持有组织地学习以及避免重复地打造解决通用问题的车轮（车轮在任何地方都被置于一个反复重用的过程）。归档是如此重要，但有用的文档最好不要冗长繁杂。我们生活在一个迅速变化的世界里，请保持文档简洁吧。过程也不要冗长繁复。CMM 关注做事（doing things），而不是成事（having things）。一个 1-2 页的过程说明是足够了，子过程和步骤能按需调用就行。运用好的软件设计原则，比如在定义过程中使用内存空间的局域性、信息隐藏以及抽象。另外的实用经验是每周最多跟踪 2-3 个任务的工作。其顺序应由少量指导性原则或法则所确立，而不是由复杂的控制来确立[Wheatley92, page 11]。

过程需要剪裁到项目所需的程度[Ginsberg95,Ade96]。虽然标准过程提供了一个基础，但每一个项目也都将有其独特的要求。剪裁时不切实际的约束可导致执行过程中的重大阻力。正如 Hoffman 所表述的，“别生造感悟也别强求过程。”[Hoffman98]

过程所需的形式化程度对大组织和小组织都构成了威胁[Comer98]。对于那些每每提到要“按照文档化步骤”的等级 2 的 25 个关键实践中的每一个，应该存有很个别的步骤吗？答案正如在 CMM 书籍《文档和 CMM》(Documentation and the CMM)的 4.5.5 小节所论述的，是一个响亮的“不!”——文档的包装是组织化的必然结果。

文档化的过程如果没有加以有效地部署，只会具有很小的价值。要想达成文档化的大量买进，过程实现必须成为过程定义和改进的一部分。通过多种机制的培训，对于协调一致和效力非凡的软件工程及管理将是建设性的。培训的动机是发展技能。有很多“培训机制”不同于能有效培养技能的正规课堂培训。其中应该被认真考虑的是正规的顾问制。在此情形下，形式化意谓着寄希望于没有经验的顾问。形式化提示要在如何指导及监督顾问的效力上训练人们。

在过程或技术的最初部署之后，培训仍然是一个问题[Abbott97，Williams98]。当人员变动时，培训所增加的需要可能不会被充分地认识。顾问学徒制能充分地解决这一问题，但是除非能谨慎地加以监督，否则不能设想会有满意的结果。

管理上的培训是特别重要的，因为低下的管理会削弱一个好的团队。那些因其技术技能而被提拔到管理层的人，将不得不重新学习包括人际关系在内的一套新的技能[Mogilensky94, Curtis95,Weinberg94]。

软件工程管理的部分论题确实是风险管理。感觉上，CMM 就是关于管理风险的。我们致力于确立稳定的需求，以便能有效地实施计划和管理，但商业环境总在迅急而紊乱地变化着。我们尝试在软件那混沌的大海里建造秩序的孤岛，但是秩序和混沌都自有其位置。Wheatley 建议，“为了生存，开放系统维持在一非平衡状态，保持系统均衡以便它能改变和成长。”[Wheatley92, page 78]尽管我们能确立帮助我们管理混沌世界里的风险的过程，我们也需要变化和成长。

这意味着你应该使用一个增量的或进化的生存周期。如果你想重心集中到风险管理上，螺旋模型将是首选的生存周期模型。如果重心集中在笼络客户上，可能快速原型法或接合应用设计更好一些。少数长期项目对稳定环境的奢求是必要的，对其瀑布生存周期将是首选——可能它仍是最普遍通用的生存周期。注意，无论如何，对于小项目，瀑布生存周期都会是一个极佳的选择。

成功的过程定义和改进的头号因素是“全程计划（planfulness）”[Curtis96]。计划是每一个主要软件过程都需要的，但不外乎绑定合理的判断、由组织决定什么是“主要的”以及怎样包装计划。一个计划可以驻留在几个不同的工件或被植入一个大计划当中。

即使你可以对最好的同行评审有争议，但简单的事实是同行评审带来的好处远远超出了其成本。数据表明一些检测表格应该是惯用的[Ackerman89]，但任何学院式或严格的评审表格，诸如结构化预排，都附加了重要价值。不幸地，认可同行评审并不意味着我们在系统地做着这件事。我们需要“走在路上”，而不仅仅是“说在嘴上”。这对技术人员来说是很大的挫折，他们不理解 CMM 所强调的管理，然而糟糕的管理会导致放弃好的工程实践，比如同行评审。

另外还有其他被确定为是小组织和小项目的问题，Paquin[Paquin98]认定了下面的 5 个：

- 评估
- 工程焦点
- 文档
- 需求功能
- 成熟度提问单

我们没有讨论等级 2 的项目焦点对小组织构成的挑战。软件过程改进包括了那些对小项目而言是过度的开销。一些劝告从组织化观点来攻击恰似混合了等级 2 和等级 3 而又的确不失为合理途径的小项目的过程改进[Comer98，Paquin98]。CMM 本就是为任何规模的组织或项目而考虑的[Paulk96]。尽管无须过程焦

点一个组织也能达到等级 2，但极其高效的组织化学习的策略将成为减少项目开销的组织资产的一个重点。同时，必须认识到项目等级的改变可能会形成多半是基于正当利害关系的阻力，而且探察阻力时需要考虑组织进行学习过程的那部分。

需求功能是一个问题，因为比起人来可能有更多的 CMM 功能。这个问题是做为技术或角色的映射来讨论的。成熟度提问单因使用 CMM 技术而成为关注焦点，它可能在填写的过程中被搞乱。以组织的技术来表达提问单，甚至对于非正式的评估及度量也是很需要的先导。

Abbott[Abbott97]指明了对于小组织的软件过程改进的 6 个关键：

- 高层管理的支持
- 正确的用人原则
- 将项目管理的法则应用到过程改进
- 与 ISO 9001 的集成
- 来自过程改进顾问的协助
- 聚焦在提供项目上及商业上的价值

如果将好的项目管理应用到软件项目中是确保成功的最佳途径，那么应该象对待其他任何项目一样来对待过程改进就同样是正确的。ISO 9001 是在大组织里比小组织里使用更频繁的一个评估版本，因而 Abbott 也有兴趣针对他的小公司指出这一点。

Brodman 和 Johnson[Johnson98]认为针对小组织/小项目的挑战有 7 种：

- 处理需求
- 生成文档
- 管理项目
- 分配资源
- 度量过程
- 促导评审
- 提供培训

Brodman 和 Johnson 开发了一个针对小型商业、组织和项目的 CMM 剪裁版本[Johnson96, Johnson97, Brodman94]。尽管如此，大多数 CMM 的关键实践还是被剪裁到了《LOGOS 剪裁版 CMM》里，变化体现在：

- 已存实践的净化
- 明显的夸张
- 选择性实践的介绍(特出地作为例子)
- 伴随小商业/小组织/小项目的结构及资源的实践调整

因此针对小组织而剪裁 CMM 的相关改变是可以根本不予考虑的。

4. 滥用 CMM

正确使用 CMM 意味着要平衡相互冲突的目标。基于 CMM 的评估要求运用专业性的判断。尽管如此，CMM 在做出这些判断以及消除对一个明确的、反复的、非设计工作特有的过程的主观臆断等方面都提供了大量的指导。CMM 有时作为一整套过程需求被提及，但它不能有任何含有“将要”(shall)的语句。这就是在核对（子）实践一致性时造成 CMM 滥用的原因。

有些是勉强地、无能地去解释、剪裁或应用判断力。这是易于委派到关键实践的，但却愚笨透顶。此种蠢行常常由客户意图及能力的偏执来驱动。我在更多的场合听说某人要做某些傻事，但他们害怕客户无知无能到理解不了不能完全生搬硬套 CMM 的道理的地步。这对于软件能力评价来说是明显有疑问的。判断可能不一致是对的——并且有时只有如此才合理。曾在某一环境中适用的却可能满足不了一个新项目。那就是我们推荐把过程成熟度包含在风险评估要比使用成熟度等级来过滤报价者更好的原因。小组织应该较少关心这个问题，因为对小组织的软件能力评价未必是划算的。它是从事许多小项目的大组织的一个问题的多个方面。

十分不幸，我没有此问题的解决方案。象这种 CMM 能帮助组织改进软件过程的“标准”，却是聚焦在达到一个没有从事潜在的、能导致紊乱行为的过程的能力成熟度等级。成熟度应该成为改进的度量，而不是改进的目标。这就是为何我们强调需要依靠改进来达到商业目的。

5. 结论

底线是软件过程改进应该有助于达成商业成就——而不是为其自身理由。无论对大组织还是小组织，这都是真真确确的。最好的忠告来自于 Bellcore 的总裁 Sanjiv Ahuja：“让常识奏效！”

构造软件是一项设计密集的创造性活动。同时过程的纪律对于能否成功是至关重要的——目标是解决问题——同时要有创造活力。就算软件本身不是重复的，软件的过程也应该是可重复的。要在纪律约束和创造活力之间取得均衡是颇具挑战性的[Glass95]。失去了创造性视野，软件工作的精密设计的天性将被引入令人窒息的僵化境地。而失去了所需的纪律观念，也将导致混乱不堪。

CMM 描绘了软件过程改进的一个“常识工程化”路径。它的成熟度等级、关键过程区域、目标及关键实践经受了软件社团内部的广泛讨论和评审。同时 CMM 既非完美无缺也不包容一切，它只是表达了软件社团的一种广泛一致的意见，并且成为指导改进工作的一个有用工具，它也有助于小型软件组织改进它们的过程 [Abbott97, Hadden98b, Hoffman98, Pitterman98, Sanders98]。

小组织应该认真地考虑 PSP 和 TSP[Ferguson97,Hayes97]。掌握了 PSP 课程，我可以高度评价其建立了自我约束的能力。注意看书的效果不能等同于掌握课程及实际操作。CMM 所从事的过程改进组织化的一个侧面，就是 PSP 致力于建造个体从业者的能力。PSP 确信个人能够——基于他或她自有的性能数据——遵从纪律尊重价值——以工程化途径来构建软件。

参考资料

Abbott97 John J. Abbott, "Software Process Improvement in a Small Commercial Software Company,", Proceedings of the 1997 Software Engineering Process Group Conference, San Jose, CA, 17-20 March 1997.

Ackerman89 A.F. Ackerman, L.S. Buchwald, and F.H. Lewski, "Software Inspections: An Effective Verification Process," IEEE Software, Vol. 6, No. 3, May 1989, pp. 31-36.

Ade96 Randy W. Ade and Joyce P. Bailey, "CMM Lite: SEPG Tailoring Guidance for Applying the Capability Maturity Model for Software to Small Projects," Proceedings of the 1996 Software Engineering Process Group Conference: Wednesday Papers, Atlantic City, NJ, 20-23 May 1996.

Austin96 Robert D. Austin, Measuring and Managing Performance in Organizations, Dorset House Publishing, ISBN: 0-932633-36-6, New York, NY, 1996.

Barbour96 Rick Barbour, "Software Capability Evaluation Version 3.0 Implementation Guide for Supplier Selection," Software Engineering Institute, Carnegie Mellon University, CMU/SEI-95-TR-012, April 1996.

Brodman94 J.G. Brodman and D.L. Johnson, "What Small Businesses and Small Organizations Say About the CMM," Proceedings of the 16th International Conference on Software Engineering, IEEE Computer Society Press, Sorrento, Italy, 16-21 May 1994, pp. 331-340.

Clark97 Bradford K. Clark, "The Effects of Software Process Maturity on Software Development Effort," PhD Dissertation, Computer Science Department, University of Southern California, August 1997.

Collins94 James C. Collins and Jerry I. Porras, Built to Last, HarperCollins Publishers, New York, NY, 1994.

Curtis95 Bill Curtis, William E. Hefley, and Sally Miller, "People Capability Maturity Model," Software Engineering Institute, CMU/SEI-95-MM-02, September 1995.

Curtis96 Bill Curtis, "The Factor Structure of the CMM and Other Latent Issues," Proceedings of the 1996 Software Engineering Process Group Conference: Tuesday Presentations, Atlantic City, NJ, 20-23 May 1996.

DeMarco95 Tom DeMarco, Why Does Software Cost So Much?, ISBN 0-932633-34-X, Dorset House, New York, NY, 1995.

DOD87 Department of Defense, "Report of the Defense Science Board Task Force on Military Software," Office of the Under Secretary of Defense for Acquisition, Washington, D.C., September 1987.

Ferguson97 Pat Ferguson and Jeanie Kitson, "CMM-Based Process Improvement Supplemented by the Personal Software Process in a Small Company Environment,", Proceedings of the 1997 Software Engineering Process Group Conference, San Jose, CA, 17-20 March 1997.

Gibbs94 W. Wayt Gibbs, "Software's Chronic Chrisis," Scientific American, September 1994, pp. 86-95.

Ginsberg95 Mark Ginsberg and Lauren Quinn, "Process Tailoring and the Software Capability Maturity Model," Software Engineering Institute, CMU/SEI-94-TR-024, November 1995.

Glass95 Robert L. Glass, Software Creativity, Prentice Hall, Englewood Cliffs, NJ, 1995.

Hadden98a Rita Hadden, "How Scalable are CMM Key Practices?" Crosstalk: The Journal of Defense Software Engineering, Vol. 11, No. 4, April 1998, pp. 18-20, 23.

Hadden98b Rita Hadden, "Key Practices to the CMM: Inappropriate for Small Projects?" panel, Rita Hadden moderator, Proceedings of the 1998 Software Engineering Process Group Conference, Chicago, IL, 9-12 March 1998.

Hayes97 Will Hayes and James W. Over, "The Personal Software Process (PSP): An Empirical Study of the Impact of PSP on Individual Engineers," Software Engineering Institute, Carnegie Mellon University, CMU/SEI-97-TR-001, December 1997.

Herbsleb97 James Herbsleb, David Zubrow, Dennis Goldenson, Will Hayes, and Mark Paulk, "Software Quality and the Capability Maturity Model," Communications of the ACM, Vol. 40, No. 6, June 1997, pp. 30-40.

Hoffman98 Leo Hoffman, "Small Projects and the CMM," in "Key Practices to the CMM: Inappropriate for Small Projects?" panel, Rita Hadden moderator, Proceedings of the 1998 Software Engineering Process Group Conference , Chicago, IL, 9-12 March 1998.

Humphrey95 Watts S. Humphrey, A Discipline for Software Engineering, ISBN 0-201-54610-8, Addison-Wesley Publishing Company, Reading, MA, 1995.

Johnson96 Donna L. Johnson and Judith G. Brodman, The LOGOS Tailored Version of the CMM for Small Businesses, Small Organizations, and Small Projects, Version 1.0, August 1996.

Johnson97 Donna L. Johnson and Judith G. Brodman, "Tailoring the CMM for Small Businesses, Small Organizations, and Small Projects," Software Process Newsletter, IEEE Computer Society Technical Council on Software Engineering, No. 8, Winter 1997, p. 1-6.

Johnson98 Donna L. Johnson and Judith G. Brodman, "Applying the CMM to Small Organizations and Small Projects," Proceedings of the 1998 Software Engineering Process Group Conference, Chicago, IL, 9-12 March 1998.

Lawlis95 Patricia K. Lawlis, Robert M. Flowe, and James B. Thordahl, "A Correlational Study of the CMM and Software Development Performance," Crosstalk: The Journal of Defense Software Engineering, Vol. 8, No. 9, September 1995, pp. 21-25. Reprinted in Software Process Newsletter, IEEE Computer Society Technical Council on Software Engineering, No. 7, Fall 1996, pp. 1-5.

McFeeley96 Bob McFeeley, "IDEAL: A User's Guide for Software Process Improvement," Software Engineering Institute, CMU/SEI-96-HB-001, February 1996.

Mogilensky94 Judah Mogilensky and Betty L. Deimel, "Where Do People Fit in the CMM?," American Programmer, Vol. 7, No. 9, September 1994, pp. 36-43.

Paquin98 Sherry Paquin, "Struggling with the CMM: Real Life and Small Projects," in "Key Practices to the CMM: Inappropriate for Small Projects?" panel, Rita Hadden moderator, Proceedings of the 1998 Software Engineering Process Group Conference, Chicago, IL, 9-12 March 1998.

Paulk95 Carnegie Mellon University, Software Engineering Institute (Principal Contributors and Editors: Mark C. Paulk, Charles V. Weber, Bill Curtis, and Mary Beth Chrissis), The Capability Maturity Model: Guidelines for Improving the Software Process, ISBN 0-201-54664-7, Addison-Wesley Publishing Company, Reading, MA, 1995.

Paulk96 Mark C. Paulk, "Effective CMM-Based Process Improvement," Proceedings of the 6th International Conference on Software Quality, Ottawa, Canada, 28-31 October 1996, pp. 226-237.

Pitterman98 Bill Pitterman, "Key Practices to the CMM: Inappropriate for Small Projects?" panel, Rita Hadden moderator, Proceedings of the 1998 Software Engineering Process Group Conference, Chicago, IL, 9-12 March 1998.

Sanders98 Marty Sanders, "Small Company Action Training and Enabling," in The CMM and Small Projects, Society for Software Quality Roundtable, Washington, DC, 26 January 1998.

Senge90 Peter M. Senge, The Fifth Discipline: The Art & Practice of the Learning Organization, Doubleday/Currency, New York, NY, 1990.

Strigel95 Wolfgang B. Strigel, "Assessment in Small Software Companies," Proceedings of the 1995 Pacific Northwest Software Quality Conference, 1995, pp. 45-56.

Weinberg94 Gerald M. Weinberg, Quality Software Management, Volume 3: Congruent Action, ISBN 0-932633-28-5, Dorset House, New York, NY, 1994.

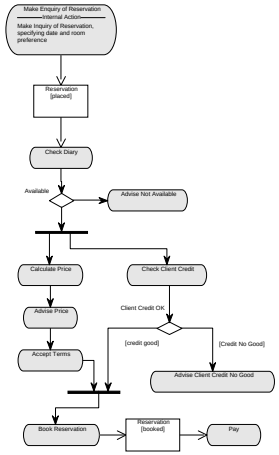
Wheatley92 Margaret J. Wheatley, Leadership and the New Science, Berrett-Koehler Publishers, San Francisco, CA, 1992.

Williams98 Louise B. Williams, "SPI Best Practices for 'Small' Projects," in The CMM and Small Projects, Society for Software Quality Roundtable, Washington, DC, 26 January 1998

.

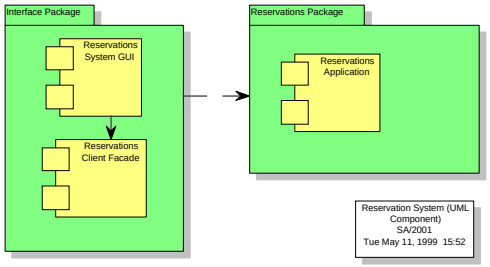
原文链接 http://www.umlforum.com/zippdf/cmm-small.zip

UML Activity Diagrams（UML 活动图），可以完整地描述企业的工作流程，即人员完成工作所需经历的步骤，能补足 Class diagram 不足之处。 它最主要的价值是去发现所有的并行程序，以排除不必要的商业程序。

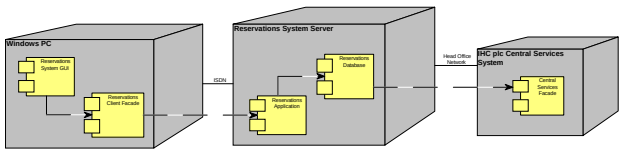


Implementation

A Component Diagram （构成图）用来塑造软体结构的模型，包含附属在软体间的元件、二进位码元件及可执行的元件。

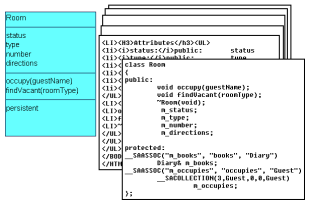


Deployment Diagrams （配置图）用来模拟处在执行状态的元素及其所涵盖的软体的元件、程序及物件的配置情况。 在这图型里，可以模拟实体结点 (Physical Nodes)及结点间沟通的联结关系。



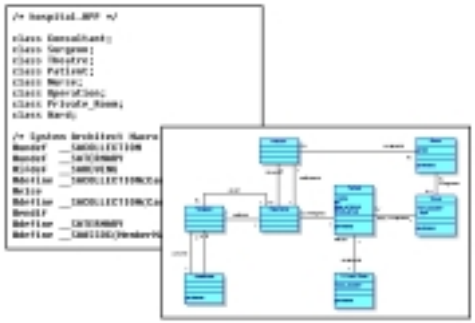
Generating Code

经由 Class Diagram 可以自动产生多种程式码；如：C++, Java, Visual Basic, HTML, CORBA IDL, Smalltalk, 及 Delphi Object Pascal.



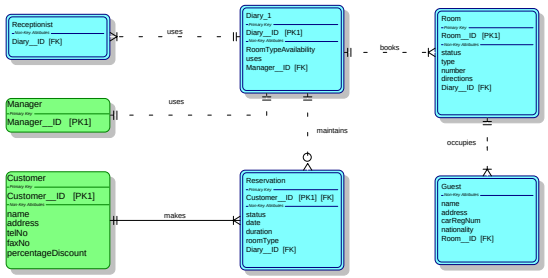
Reversing Code

System Architect 2001 可提供 C++ 及 Java 程式码的逆向工程。



Generating an ERD

您可以利用 UML Class diagram 自动产生 Entity Relation Diagram（实体关系图）。Classes （类别）的设计是与 Entity（实体）相对映。Class Attributes （类别属性）与实体属性相对映。



System Architect 2001

UML Object Oriented

物件导向系统

简 介

POPKIN SOFTWARE

戊丰科技股份有限公司
台北市敦化南路一段 21 号 4 楼之二
Tel：(02)2570-1877 Fax：(02)2570-1876

Methodology Outline

System Architect 2001 支援数种物件导向的方法论，包括： OMT/Rumbaugh, Booch, Shlaer/Mellor, and Coad/Yourdon，另也支援 Jacobson Use Case Diagram 以及 UML。

The **Unified Modeling Language** (UML).

Unified Modeling Language（统一塑模语言）是一个标示物件导向系统分析和设计的发展标准。 System Architect 2001 支援 UML 1.3 的规格。

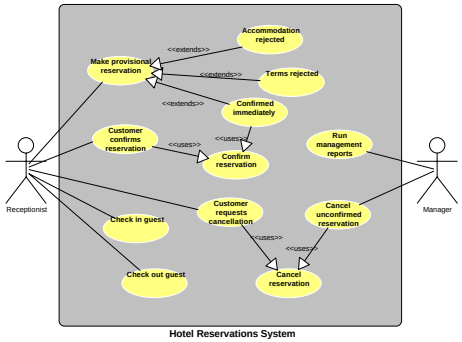
虽然 UML 并不是规定一个发展物件导向的标准程序或方法，但它是指定一个标准的图型及标记的组合，强迫使用者塑造确切的系统模型。

System Architect 2001 Features

- 32 位元使用者界面，及完整的企业模型整合工具。
- 多合一的电脑辅助设计工具（CASE tool ），包含商业描述、物件导向、元件设计及资料模型。
- 支援的法论包括： UML, IDEF0, IDEF3, IDEF1X, Yourdon, Ward-Mellor, Gane & Sarson, SSADM, Catalyst, OMT, Booch and Shlaer/Mellor
- 支援多人使用的网路版本，整合 Microsoft VBA、互动式关联矩阵(Matrix)及影响分析。
- 可以产生 Microsoft Word 文件及 HTML 网页文件。
- C++, Java, Dynasty & Magic – Generation & Reversing, HTML Generation, Simulation.

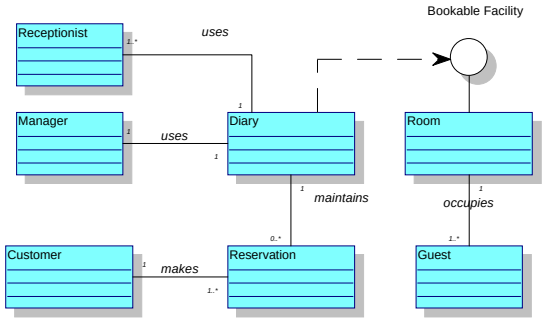
Business-Level Analysis

Use Case 技术可以获得记录该系统或企业目前所做的事及应该做的事**的资讯**。Use Case 技术可帮助您建立系统作业流程的架构模型。它是一个通向物件导向系统分析的绝佳方法。



Static Analysis & Design

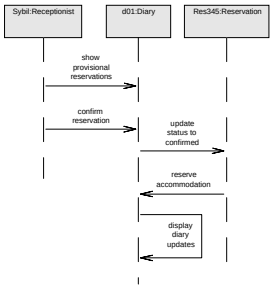
The Class Diagram 是一个系统主要的静态分析图，在这图型里，利用类别和继承的关系，释明了系统的类别结构。



System Architect 2001 支援物件间沟通的介面符号，但沟通介面无法执行(Implementation)，它们缺少属性(Attributes)、状态(States)或关联(Association)；只有操作的方法(Operations)。一个沟通介面（Interface）可能以印有"Interface"的矩形类别（Class）符号显示，或是以类似棒棒糖的符号来表示。

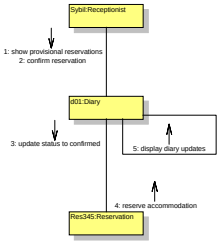
Dynamic Analysis & Design

UML Sequence Diagram 是用来塑造系统架构执行（Implementation）细节的模型。UML Sequence Diagram 表达出物件间交换的讯息(Message) 以产生期望的结果。



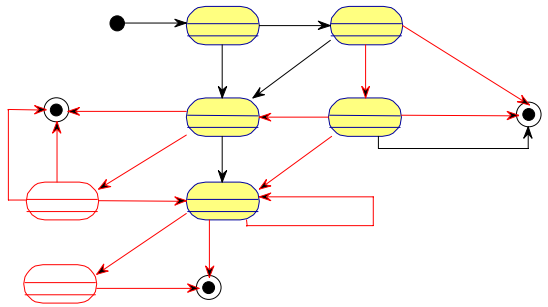
两个物件间沟通的讯息(Message)，由被呼叫的方法(Method Called)所定义；它是由传送的物件(Sending Object)指向接收物件(Receiving Object)；被呼叫的方法(Method)必须属于接收物件(Receiving Object)中所定义的类别实例(Instance of Class) 。

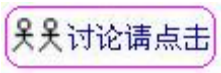
UML Collaboration Diagram （UML 联合图）是描绘物件间如何交互影响的模型架构。System Architect 2001 会自动同步给予 Collaboration Diagram 及 Sequence Diagram 相同的图型名称。



State Diagram（状态图） 是描绘系统运转的状态。 状态图是描绘系统内物件的每一个类别的典型， 包含重要的动态变化。

状态图（State Diagram）能利用父子图型的连结（Parent/Child Link）附属于类别图（Class Diagram）里的类别符号（Class Symbol）。





项目管理规范-RUP 管理实施

（第三部分）

李杰(poorjim@sina.com)

第一部分：项目阶段
第二部分：核心工作流程
第三部分：角色划分
第四部分：目前实施项目规范的考虑

概述

软件开发的产品质量水平，是一个由来已久的话题。而提高软件企业的产品质量水平，必须改进软件产品的开发过程。但是这里没有什么百试百灵的灵丹妙药，我们必须根据本企业的实际情况，参考国内外先进企业的经验，总结出一种适合本企业的软件开发模式。

此规范是基于 CMM 模型规范，以 RUP 软件工程过程为蓝本，由我本人根据项目实际情况而选择修改，从而使之适应当前应用级系统设计开发的需要。

本文主要以 RUP 的软件工程框架为主，省略复杂概念部分。着眼点放在控制软件产品开发流程上，由于人员配置与软件分工现行状况的限制，对其中的部分细节进行了合并可省略，从而适应目前国内软件开发所要求。

Rational Unified Process（简称 RUP）是一套软件工程过程（在下面介绍）。

在 RUP 过程中，我们可以看到它非常强调一点：**循环**。

现在我们做的每一个项目都存在不断变化的问题。用户需求变化、系统设计变化（可能是需求变化也可能是存在了技术问题）、编码变化（由测试与复审等环节引发的）等问题困扰着项目进行。解决这些问题的方法就是不断的循环。

这个规范是我根据自己的观点整理编写而成的，有不足之处请指教。

李杰

2001 年 3 月 14 日

3. 角色划分

角色是抽象的职责定义，它定义的是所执行的一组**活动**和所拥有的一组**文档与模型**。

角色通常由一个人或作为团队相互协作的多个人来实现。

项目团队成员通常要履行许多不同的角色职能；就象一个人可以担任许多职务，一个人也可以担任许多不同的角色。

角色并不代表个人，而是说明个人在业务中应该如何表现以及他们应该承担的责任。

分析员角色集

分析员角色集用于组织主要从事需求获取和研究的各种角色。

角色

- 业务流程分析员
- 业务设计员
- 业务模型复审员
- 需求复审员
- 系统分析员
- 用户界面设计员

3.1.1. 业务流程分析员

业务流程分析员通过概括和界定作为建模对象的组织来领导和协调业务用例建模。例如，确定存在哪些业务主角和业务用例，他们之间如何进行交互。

人员配备

担任业务流程分析员的人员应该善于简化工作，并且具有良好的沟通技巧。担任此角色的人员中必须要有具备业务领域知识的人才，但这种知识并不是所有人都必备的。

业务流程分析员应该准备好开展以下工作：

- 评估将在其中部署项目最终产品的目标组织的情况。
- 了解客户与用户的需求、策略和目标。
- 协调目标组织的建模工作。
- 在必要时对业务工程工作进行讨论和协调。

- 对目标组织中所建议的任何变更进行成本效益分析。

3.1.2. 业务设计员

业务设计员通过描述一个或几个业务用例的工作流程来详细说明组织中某一部分的规约。他指定实现业务用例所需的业务角色及业务实体，并且将业务用例的行为分配给这些业务角色及业务实体。业务设计员定义一个或几个业务角色和业务实体的责任、操作、属性和关系。

人员配备

担任业务设计员的人员应该善于协调，并且具有良好的沟通技巧。他最好具有业务领域的知识，但这并不是担任此角色的所有人都必需的。业务设计员需要熟悉用于获取业务模型的工具。

3.1.3. 业务模型复审员

业务模型复审员参与对业务用例模型和业务对象模型的正式复审。

人员配备

在大多数情况下，担任业务模型复审员的人员都需要具备业务领域的基本知识，或者对将用来实现业务自动化的技术具备基本的知识。业务模型复审员应该具备的另一种技能是详细了解所应用的业务工程技术。

3.1.4. 系统分析员

系统分析员通过概括系统的功能和界定系统来领导和协调需求获取及用例建模。例如，确定存在哪些主角和用例，以及他们之间如何交互。

人员配备

担任系统分析员的人员应该善于协调，并且具有良好的沟通技巧。担任此角色的人员中必须要有具备业务和技术领域知识的人才，但这些知识并不是所有人都必须具备的。

3.1.5. 用户界面设计员

用户界面设计员通过以下方法领导和协调用户界面的原型设计和正式设计：

- 分析对用户界面的需求，包括可用性需求；
- 构建用户界面原型；
- 邀请用户界面的最终用户参与可用性复审和使用测试会议；
- 对用户界面的最终实施方案（由设计员和实施员等其他开发人员创建）进行复审并提供相应的反馈。

人员配备

用户界面设计员不应实施用户界面。用户界面设计员的工作重点和时间都应集中在用户界面的设计和“可视化成形”，原因如下：

- 用户界面设计员所需的技能通常需要为当前的项目和应用程序类型（可能具有独特的可用性需求）而加以改进和优化，这需要投入时间并集中工作重点。
- 应该限制因“一心二用”而带来的风险，即用户界面设计员不应该因为实施方面的考虑（相对于可用性方面的考虑而言）而受到过多的影响。

3.2. 开发角色集

开发人员角色集用于组织主要从事软件设计与开发的各种角色。

角色

- 构架设计师
- 构架复审员
- 代码复审员
- 数据库设计员
- 系统设计员
- 设计复审员
- 实施员
- 集成员

3.2.1. 构架设计师

构架设计师负责在整个项目中对技术活动和工件进行领导和协调。构架设计师要确立每个构架视图的整体结构：视图的详细组织结构、元素的分组以及这些主要分组之间的接口。因此，与其他角色相比，构架设计师的见解重在广度，而不是深度。

人员配备

构架设计师必须多才多艺、成熟练达、洞察力强、经验丰富。这样，他才能在无法获得完整信息的情况下迅速领会问题并根据经验作出审慎的判断。更准确地说，构架设计师（或者构架团队的成员）必须兼具以下技能：

- **经验**：既包括在问题领域的经验（通过彻底了解需求），也包括在软件工程领域的经验。对于一个构架团队，这些素质要求可由各团队成员来分别承担，但其中至少要有一名构架设计师能够把握项目

的全局。

- **领导才能**：能够推动各个团队的技术进展，并能在压力下作出关键性的决策然后将其贯彻到底。要提高效率，构架设计师和项目经理必须紧密协作。构架设计师主要负责解决技术问题，项目经理主要负责解决行政管理问题。构架设计师必须有权在技术问题上作出决定。
- **沟通**：能够赢得他人的信任，以对其进行说服、激励和指导。构架设计师不能靠命令进行领导，而必须要赢得项目中其他人员的赞同。为了提高效率，构架设计师必须赢得项目团队、项目经理、客户、用户群体以及管理团队的尊敬。
- **以目标为中心、积极主动**，不懈地追求成效。构架设计师是推动项目发展的技术动力，而不是空想家。在其职业生涯中，成功的构架设计师一直都要在捉摸不定和承受压力的情况下作出折衷决定。构架设计师只有将注意力集中在该做的事情上，才能在项目中取得成功。

从专业角度看，构架设计师必须具备系统设计员的所有能力。

团队。如果项目较大，需要组建一个构架团队，则应尽量广聚贤才，使该团队既拥有广泛的经验，又对软件工程流程具有一致的认识。构架团队不应该是由各团队、领域或承包商的代表组成的委员会。软件构架设计是一项长期的工作，始终都需要配备专职人员。

3.2.2. 构架复审员

一般而言，**构架复审员**负责计划并执行对软件构架的正式复审。

人员配备

构架复审员角色的人员配备要求与**构架设计师**的人员配备要求相同，但前者更加注重于技术问题。虽然对领导才能、成熟程度、实用主义及注重结果这些方面的重视程度稍低，但这些方面仍然重要：复审员可能会发现构架方面的缺陷，并且有可能会因为影响项目的进度而不受欢迎。尽管如此，最好还是在问题可以解决的时候及早提出关键性的问题，而不是盲目地追随进度，致使项目团队步入歧途。构架复审员需要根据成本对风险加以权衡，并对影响项目成功的概括性问题保持一定的敏感性。构架复审员还需是善于说服的沟通者，他应该能够提出并讨论对他人来说比较敏感的问题。

3.2.3. 代码复审员

代码复审员负责确保源代码的质量，并且计划和执行源代码复审。在复审活动中，代码复审员还负责有关返工的任何反馈意见。

3.2.4. 数据库设计人员

数据库设计员定义表、索引、视图、约束条件、触发器、存储过程、表空间或存储参数，以及其他在存储、检索和删除永久性对象时所需的数据库专用结构。相关信息记录在数据模型中。

人员配备

数据库设计员必须在以下方面具有扎实的应用知识：

- 数据库和面向对象的分析设计技术
- 系统构架，包括数据库和系统性能调整，以及硬件和网络负载平衡
- 数据库管理
- 了解实施语言和环境

3.2.5. 系统设计员

设计员定义一个或几个类的职责、操作、属性及关系，并确定应如何根据实施环境对它们加以调整。此外，设计员可能要负责一个或多个设计包或设计子系统，其中包括设计包或子系统所拥有的所有类。

人员配备

设计员必须在以下方面具有扎实的应用知识：

- 用例建模技术。
- 系统需求。
- 软件设计技术，包括：
  - 面向对象的分析设计技术。
  - 统一建模语言。
- 实施系统时将利用的技术。

3.2.6. 设计复审员

设计复审员计划并进行设计模型的正式复审。

人员配备

设计复审员的人员配备要求与构架设计师的人员配备要求相同，但前者更加侧重于技术问题。虽然对领导才能、成熟程度、实用主义及注重结果这些方面的重视程度稍低，但这些方面仍然重要：复审员可能会发现设计方面的缺陷，并且有可能会因为影响项目的进度而不受欢迎。尽管如此，最好还是在问题可以解决的时候及早提出关键性的问题，而不是盲目地追随进度，致使项目团队步入歧途。设计复审员需要根据风险对成本加以权衡，并对影响项目成功的概括性问题保持一定的敏感性。设计复审员还需是一个善说服的沟通者，他应该能够提出并讨论对他人来说比较敏感的问题。

从技术知识的观点来看，设计复审员应该具有与设计员相同经验。

3.2.7. 实施员（程序员）

实施员负责按照项目所采用的标准来进行构件开发与测试，以便将构件集成到更大的子系统中。如果必须创建驱动程序或桩模块等测试构件来支持测试，那么实施员还要负责开发和测试这些测试构件及相应的子系统。

人员配备

实施员应具备的相应技能和知识包括：

- 了解系统或所测试的应用程序
- 熟悉测试及测试自动化工具
- 编程技能

建议负责实施子系统的实施员同时应负责该子系统所包含的构件。

3.2.8. 集成员

实施员将经测试的构件交付到集成工作区，由集成员在集成工作区将构件组合起来，生成一个工作版本。集成员还负责制定集成计划。集成在子系统和系统级别进行，每次集成均有独立的集成工作区。正如经测试的构件从实施员的专用开发工作区交付到子系统集成工作区一样，已集成的实施子系统也从子系统集成工作区交付到系统集成工作区。

人员配备

有时，担任集成员的个人还可以担任实施员或测试员。例如，如果项目较小，或者是在子系统级别上进行集成，就可以让同一个人兼任集成员和测试员，以做到有效地利用人力资源。实际上，对于子系统级别的集成（和测试），一个人就可以兼任实施员、集成员和测试员的角色。但是，对于系统级别的集成，建议应由独立的团队来执行集成和测试。

3.3. 测试员角色集

测试员角色集用于组织主要从事软件测试的各种角色。

角色

- 测试设计员
- 测试员

3.3.1. 测试设计员

测试设计员是测试中的主要角色。该角色负责对测试进行计划、设计、实施和评估，包括：

- 生成测试计划和测试模型
- 执行测试过程
- 评估测试范围和测试结果，以及测试的有效性
- 生成测试评估摘要

人员配备

测试设计员应具备的相应技能和知识包括：

- 了解系统或所测试的应用程序
- 了解测试及测试自动化工具
- 具备诊断和解决问题的技能
- 编程技能（最好具备）

3.3.2. 测试员

测试员负责执行测试，其职责包括：

- 设置和执行测试
- 评估测试执行过程并修改错误

人员配备

测试员应具备的知识和技能可能会因为他们所执行的测试类型和/或测试阶段的不同而有所差异。例如，在执行性能测试或集成阶段的测试时，需要更高级的技能。在执行功能测试或系统测试阶段的测试时，则不需要太高级的技能。

以下是测试员所需知识和技能的一些标准：

高级测试员：

- 了解系统或所测试的应用程序
- 了解联网和系统构架
- 了解测试及测试自动化工具
- 具备诊断和解决问题的技能
- 编程技能（必备）

初级测试员：

- 了解系统或所测试的应用程序
- 了解测试及测试自动化工具
- 具备诊断和解决问题的技能
- 编程技能（最好具备）

3.4. 经理角色集

经理角色集用于组织主要从事软件工程流程的管理与配置的各种角色。

角色

- 变更控制经理
- 配置经理
- 部署经理
- 流程工程师
- 项目经理
- 项目复审员

3.4.1. 变更控制经理

变更控制经理这一角色负责对变更控制过程进行监督。此角色通常由配置（或变更）控制委员会 (CCB) 来担任，该委员会应该由有关各方（包括客户、开发人员和用户）的代表组成。在小型项目中，项目经理或软件构架设计师一人即可承担此角色。

3.4.2. 配置经理

配置经理负责为产品开发团队提供全面的配置管理 (CM) 基础设施和环境。CM 的作用是支持产品开发行为，使开发人员和集成员有适当工作区来构建和测试其工件，并且使所有工件均可根据需要包含在部署单元中。配置经理还必须确保 CM 环境有利于进行产品复审、更改和缺陷跟踪等活动。配置经理还负责撰写 CM 计划并汇报基于“变更请求”的进度统计信息。

3.4.3. 部署经理

部署经理负责制定向用户群体发布产品的计划，并将其纳入布署计划中。

3.4.4. 流程工程师

流程工程师对软件开发流程本身负责。其职责包括在项目开始前配置流程，并在开发工作过程中不断改进流程。

人员配备

担任流程工程师的人员需要具有广博的软件开发知识。良好的沟通技巧是对担任此角色的人员的基本要求。

3.4.5. 项目经理

项目经理负责分配资源，确定优先级，协调与客户和用户之间的沟通。总而言之，就是尽量使项目团队一直集中于正确的目标。项目经理还要建立一套工作方法，以确保项目工件的完整性和质量。

3.4.6. 项目复审员

项目复审员负责在项目生命周期中的主要复审点处评估项目计划工件和项目评估工件。在这些复审点发生的是非常重要的复审事件，因为在它们所标志的时间点处，如果计划不够充分或者进展无法令人满意，项目很可能会就此取消。

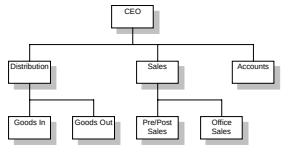
人员配备

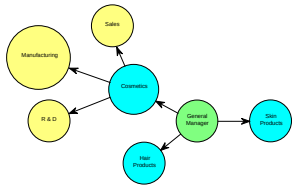
项目复审员应具有多年的业务（包括合同制订及谈判）、技术和软件项目管理的经验，并具有业务管理级别的决策能力。项目复审员还应对风险管理原理具有出色的理解力，并且善于在信息不全或不明确的环境下进行评估。

Organisation

组织模型提供一个可浏览组织的现况及未来的功能，包括组织文化、劳动力结构及能力。

Organisation charts 组织图能说明组织报告的关系及各组织单元之间的关系。





Decision Making Charts 决策制定图是企业高层视各组织单元间的关联，做为下决策时所依循的参考模式。

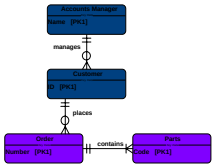
Location

The Location view 是描述商业场所的型态，一般场所及实体的场所。也可能是描述在某建筑物里的场所。

The **Process / Location Type matrix** 显示某个作业程序在那个场所型态里被完成的。

Data Modelling

System Architect 2001 利用实体模型（Entity Modelling）来描述实体、属性及实体间的关系。

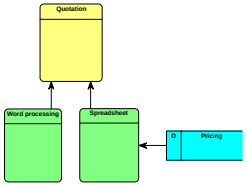


Application

Capture details of the current and future applications software required to support the business functions.

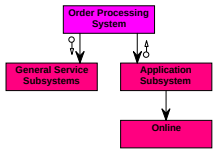
Application Area Map - high-level graphical representation of current applications.

Application Architecture Diagram - 企业高层描述欲建构的应用系统以满足商业的需求。



Application Context Diagram - high-level data flow diagram showing an application with its interfaces to existing or new computer systems.

Subsystem Structure Diagram - decomposition of a subsystem into programs.



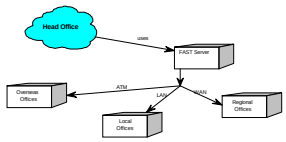
Logical Process Hierarchy - uses an object-oriented approach to form mapping between Business processes and UML.

unified modeling language

Flow Diagram - free-form flow of information or control at subsystem level.

Technology

Network Concept Diagram – 让企业高层了解企业整体的网路概况。



System Architect 2001

Business Process Modelling

Quick Reference

POPKIN SOFTWARE

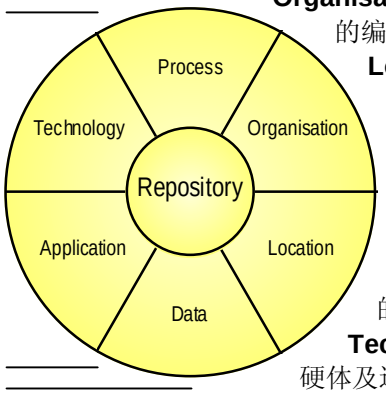
戊丰科技股份有限公司
台北市敦化南路一段 21 号 4 楼之二
Tel：(02)2570-1877　Fax：(02)2570-1876

Methodology Outline

System Architect 2001 支援 企业模型架构（**Enterprise Modelling Framework**） 让您能轻易地驾驭企业的改变及塑造资讯系统模型。

System Architect 2001 支援六个主要的产业元件，任何公司能在完成商业流程分析及设计的过程中都应该考虑到这六个元件。

Process 定义企业要做什么及如何做。



Organisation 定义企业内人员的编制。

Location 定义企业营运的场所。

Data 定义组织内应被记录的商业资料规则。

Application 详细描述软体支援企业结构的功能 。

Technology 说明企业内硬体及通讯设备。

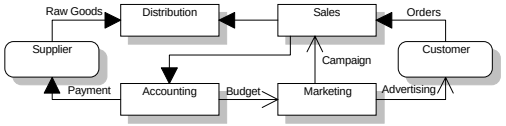
System Architect 2001 Features

- 32 位元使用者界面，及完整的企业模型整合工具。
- 多合一的电脑辅助设计工具（CASE tool ），包含商业描述、物件导向、元件设计及资料模型。
- 支援的法论包括： UML, IDEF0, IDEF3, IDEF1X, Yourdon, Ward-Mellor, Gane & Sarson, SSADM, Catalyst, OMT, Booch and Shlaer/Mellor
- 支援多人使用的网路版本，整合 Microsoft VBA、互动式关联矩阵(Matrix)及影响分析。
- 可以产生 Microsoft Word 文件及 HTML 网页文件。
- C++, Java, Dynasty & Magic – Generation & Reversing, HTML Generation, Simulation.

Business Process

企业流程模型（Business Process Model）提供所有流程及功能的分析，也提供基层作业流程分析，并显示出 动流程及必须的流程，使作业流程能完成工作。

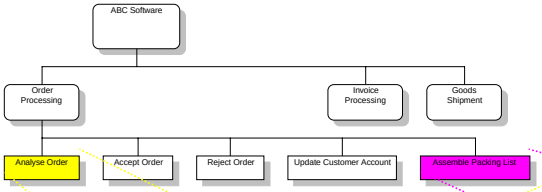
Relationship maps 显示功能间或群组间的相互影响。



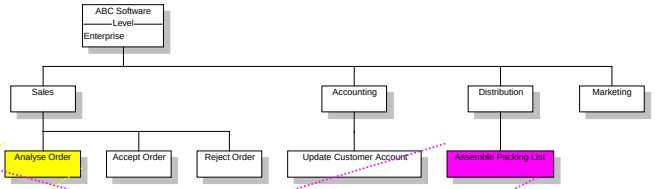
一个 **EBP(Elementary Business Process)** 是商业活动的最小单元：

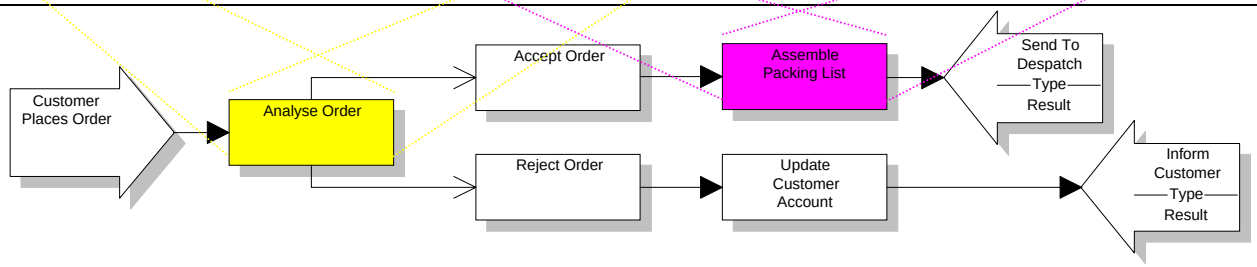
- Is worth examining
- Is performed by or in support of a single user
- May include both manual or automated activities
- Has its user involvement occur at one time

Process Oriented Hierarchy Diagrams 此图是将作业流程依流程群组作分解。

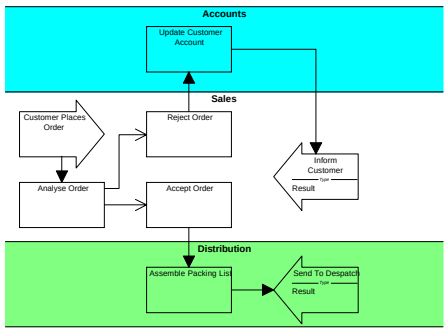


Functional Hierarchy Diagrams 此图是将作业流程依功能群组作分解。



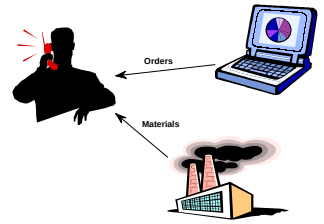


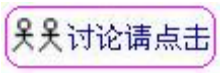
Process Chart 显示一连贯的作业流程。包含： Events, Results, Iterations, Flow Breaks, optionality and exclusivity. **Process Chart** 表示整个或一部份的 Process Thread。



Process Map 规划个别的作业程序应归属于那个组织单位内被完成。

Business Concept 以类似卡通的图案提供企业高层了解作业流程如何工作。





威赛儿商务通系统开发员手册

苏康胜

一、 前言

为了保证商务通系统项目开发成本优化并且有利于将来的扩展及重用，项目成员必须严格遵守以下开发手册。力求达成一种默认的开发规范。

二、 设计约定

- 系统将采用 RDS 技术访问远程组件，必须记住 RDS 不支持对象的属性的这一限制。
- 系统必须提供统一灵活的错误处理机制，统一管理组件的返回信息。
- 系统组件尽量遵守松耦合的原则，使组件在能重用的基础之上还能适应今后更多方面的变更。
- 为了使系统具有很强的移植性，设计要排除对使用存储过程实现的依赖等。
- 组件的设计应该避免产生过多的创建数据库连接请求。
- 详细设计必须详细到每个组件及方法和接口。
- 采用 Rational Rose 做为系统分析设计的辅助工具。
- 设计必须配合项目管理，辅助需求管理人员提供需求基线依据。
- 必须及时获取程序员反馈信息，修补设计时的漏洞。
- 其他补充可以找苏康胜商量。

三、 程序规范

a)VB 的编程规范

在软件开发过程中，编程的工作量是相当大的，同一项目参与编程的人可能有各自编程的经验和习惯，不同风格的程序代码带来了维护工作量的增加，因此为了提高代码的可读性、系统的稳定性及维护和升级的成本，程序的代码必须严格遵循统一的编程规范。

总则：

应有良好的、尽可能一致的编程风格

编码应该是严谨的、可读性强、目标明确及直观的。

注释：

无论是用户端表示层程序还是组件程序，注释必不可少。要求能占程序总量的 20%，另外注释必须在程序改变时实时更新。

每个构件的顶部应该有注释

包括：模块名称

功能描述

设计　　作者

构件内的每个过程或方法应该有注释

包括：功能描述

参数说明

返回值说明

作者　　更新

创建日期　　最后更新日期

每个模块级变量必须给出注释

重要的变量应该给出注释

描述变量的用途

变量、常量、对象的命名规约

命名必须使用大小写结合（VB 编辑器会自动转换以减少程序出错的机率）

变量命名采用[范围前缀][数组前缀][类型前缀]+[自定义命名]

控件命名采用[控件前缀]+[自定义命名]

变量范围做前缀

范围	前缀	例子
全局变量	g	gStrUserName
模块级	m	mStrUserName
过程级	无	StrUserName

数组前缀： a

类型前缀：

数据类型	前缀	例子
Boolean	Bln	BlnFound
Byte	Byt	BytRasterDate
Currency	Cur	CurBalance
Date	Dtm	DtmBeginDate
Double	Dbl	DblFee
Integer	Int	IntQty
Long	Lng	LngVcID
Single	Sng	SngAverage
String	Str	StrItemId
Object	Obj	ObjRmtsvr

ADODB.Recordset	Rst	RstItem
ADODB.Connection	Cnn	cnnNewsPaper
ADODB.Command	Cmm	CmmAddCustomer
Variant	Vnt	VntCheck
自定义类型	Udt	UdtUserInfo

控件类型命名前缀

控件类型	前缀	例子
ADO Data	ado	AdoBiblio
Check box	chk	chkReadOnly
Combo box, drop-down list box	cbo	CboEnglish
Command button	cmd	CmdExit
Common dialog	dlg	dlgFileOpen
Data-bound combo box	dbcbo	dbcboLanguage
Data-bound grid	dbgrd	dbgrdQueryResult
Data-bound list box	dblst	dblstJobType
Data combo	dbc	DbcAuthor
Data grid	dgd	DgdTitles
Data list	dbl	dblPublisher

Directory list box	dir	DirSource
Drive list box	drv	DrvTarget
File list box	fil	FilSource
Form	frm	FrmEntry
Frame	fra	fraLanguage
Graph	gra	GraRevenue
Grid	grd	GrdPrices
Horizontal scroll bar	hsb	HsbVolume
Image	img	ImgIcon
Image combo	imgcbo	imgcboProduct
ImageList	ils	IlsAllIcons
Label	lbl	lblHelpMessage
Line	lin	LinVertical
List box	lst	lstPolicyCodes
ListView	lvw	lvwHeadings
Menu	mnu	mnuFileOpen
Month view	mvw	MvwPeriod

MS Chart	ch	chSalesbyRegion
MS Flex grid	msg	MsgClients
MS Tab	mst	MstFirst
Option button	opt	OptGender
Picture box	pic	PicVGA
ProgressBar	prg	prgLoadFile
Remote Data	rd	RdTitles
Slider	sld	SldScale
Spin	spn	SpnPages
StatusBar	sta	staDateTime
SysInfo	sys	SysMonitor
TabStrip	tab	TabOptions
Text box	txt	txtLastName
Timer	tmr	TmrAlarm
Toolbar	tlb	TlbActions
TreeView	tre	treOrganization
UpDown	upd	updDirection

Vertical scroll bar	vsb	VsbRate

自行开发 ActiveX 控件的前缀

根据具体项目的设计时规定。

自定义命名空间规约

采用大小写结合的英文缩写，要求尽量能顾名思义。

代码约定：

风格约定

每个变量必须显式定义。在每个模块开始前加代码：Option Explicit。

采用缩进的格式保持程序循环的层次结构和可读性。

表示层信息对话框的图标规则

vbExclamation ------ 警告

vbQuestion ------ 询问性警告

vbInformation ------ 提示信息

vbCritical ------- 程序或系统错误

对话框的标题为模块名称（ActiveX 控件中定义一个模块极变量表示模块名称）

限制约定

模块初始化的方法统一用 Init 作为方法名称。

为了保持开发过程中与 Windows 98 环境下的兼容性，对 ADO 的引用采用（Microsoft ActiveX Object Library 2.1）,Windows 2000 下最新的是 2.5 版本的。另外也可以减少很多网络传输量。

程序开发阶段禁止使用 On Error Resume Next 语句，以免造成对错误的忽略。

程序维护前应该屏蔽所有 On Error Resume Next 语句。

程序中对数据库的操作语句只能用标准的 SQL 语句，如在 SQL SERVER 中虽然支持 delete [TABLE]。。。。，但必须使用 delete from [TABLE]。。。。保证系统的可移植性。

其它约定

为了保证程序实现正确的意图，程序编制开始前程序员应该对程序的意图有清晰、明确的理解，有任何疑问必须找系统设计人员洽谈。

b)ASP 的编程规范

略

四、 用户界面

a)概述

商务通系统将采用 B/S 结构，在 Windows 平台下开发，界面主要涉及的技术有 VB、ASP、HTML。菜单将主要采用 HTML 形式、复杂的输入输出界面将采用在 HTML 页面中嵌入 VB 的 ActiveX 控件的方式实现。查询将采用 ASP 方式实现。

b)总体规范

系统主页要求美观大方，能提供报刊亭、客户、中心职员的登录界面，登录后的页面能体现系统的总体功能框架。ASP 与 VB 的 ActiveX 控件的字体及色彩基调应该保持一致风格。

c)HTML 页面细则

- HTML 页面主要用于实现系统“主菜单”的功能，要求美观大方，利于扩展。
- 主页面上应该有公司或产品标志。
- 基本色调建议采用能反映系统特色色彩。
- “菜单”页面所有功能按钮能在一个页面的视觉范围内全部显示出来，不需要拉动滚动条。
- 页面设计过程中的基本图形元素应该按目录保存下来，以便将来功能扩展时快速修改页面。
- 查询页面的提交表单、ASP 页面字体和色彩基调应该与 ActiveX 控件保持一致。
- 密码输入要用“*”屏蔽。
- 修改 IIS 的错误返回的页面，提示用户打开浏览器选项中的容许 ActiveX 控件等信息。

- 页面提供 ActiveX 标题的显示。
- 防止特殊字符：< > < / > ‘ “对数据库提交或浏览器的影响，必须做特殊处理。
- 含有 ActiveX 控件的页面应该使控件开始获得焦点，避免控件上快捷键和浏览器上快捷键产生冲突。

d)ActiveX 规范

复杂的录入界面采用 ActiveX 控件方式实现，ActiveX 控件的界面应遵循以下原则：

1. 易用性原则
   - 控件的名称或标题最好使用用户熟悉的字眼，而且不能超过 4 个字。
   - 完成相同或相近功能的按妞用 Frame 框起来，按钮要支持快捷方式。
   - 完成同一功能或任务的元素在集中位置，减少鼠标移动的距离。
   - 按 TAB 键自动切换的顺序应该与界面上控件的排列顺序保持一致。
   - 按回车键能自动进入 TAB 顺序表示的下一控件的焦点状态（验证不通过除外）。
   - 界面上首先要输入的和重要的信息控件应该是 TAB 键顺序靠前而且在比较醒目的位置，使对应控件在一开始就获得焦点。
   - 同一界面的控件数最好不要超过 10 个，多于 10 个可以考虑按功能或属性分类的多个页面显示。
   - 分页界面的页面间用 Ctrl+Tab 的组合快捷键切换，顺序按重要程度和访问几率排列。
   - 复选框和选项框按选择几率的高低先后排列，并且有默认的选项。
   - 选项数为 2~5 时可用选项框，选项数为 2~20 时可用下拉框，大于 20 时用小查询按钮查询选择。
   - 选项数不定且数量不多的情况选用下拉框。选项数固定且数量范围为 2~5 的情况下根据界面的当前排列情况选择选项框或下拉框。
   - 按“新增”或“修改”按妞后未存盘而是按“取消”按钮应使界面回到初始状态。
   - 对显示有当前记录资料的界面按“新增”操作时，可以提示保留当前值新增和清屏新增两种方式，也可以选择取消回到初始界面。
   - 在新增和修改状态下关闭窗口或按退出按钮提示是否保存修改。

- 对于定单编号、报刊代号等的输入可以通过小按钮查询选择。

2. 安全性原则

- 可写的控件的值改变时应该提供输入验证，不合格后应该自动获得焦点。
- 首要排除可能会使应用非正常终止的错误，如：除零运算等，死循环、长度超过数据库规定的长度、含有特殊字符等。
- 应当注意排除无意的无效数据录入，如数值形态的控件要防止录入字符等。
- 应该避免未授权功能的使用及无意义的操作。
- 对可能引起致命错误或系统出错的输入字符和动作要加以限制和屏蔽。
- 界面上操作数据库中关联性强的表时，如：一对一或多的表的关系时，存储动作应该放在一个按钮中完成。
- 影响数据库操作的特殊字符，如：’、“、空格等要做特殊处理后再参与数据库操作。
- 日期采用统一的 YYYY/MM/DD 格式表示。而且日期的合理性必须得到验证。（若给大陆以外的客户使用，日期处理更加复杂）。
- 小数点的位数必须根据数据库字段的要求加以限制。
- 费率等数值不能录入负数值。
- 对输入的字符串和数值数据必须限制长度不能超过数据库规定。
- 界面能捕获 ESC 键的使用，执行退出代码的功能。
- 取消操作能回到初始状态。

3. 合理性原则

- ActiveX 控件应放在浏览器页面比较显眼的位置，能在浏览器常用工具栏显示的情况下，可以在一个屏幕中显示完整的 ActiveX 控件界面，不需拉动滚动条。
- 非法输入和操作应该有足够明确的提示说明，提示说明不能混淆和重复，如“不能输入大于 600 的数字”“数字应该小于或等于 600”应该只保留其中一种说法。不能出现 A 处有 A 的说法，B 处有 B 的说法，其实是一个意思。
- 一些键盘或鼠标的屏蔽性的操作无须用 MsgBox 来显示提示信息，如控件只能输入数字、小数点、“-”，当输入字符时不需要有任何提示。
- 提示或警告信息必须具有向导性，能告知用户错误原因，并自动使需改变的控件在第一时间内获

得焦点。

- 按钮的排列应该遵循新增、修改、删除、打印、退出等主题的排列顺序。注意界面按钮的屏蔽逻辑，如按下“新增”后屏蔽新增、修改、删除按钮，在对应位置显示存盘、取消按钮等。整个系统应该采用统一的界面按钮逻辑风格。
- 不需要另外界面输入打印查询条件的打印功能应该有预览和打印两个按钮。预览排在前面。
- 删除的动作必须先有提示信息确认是否真正删除。
- 如果执行的处理时间比较长（可能比较长）要有进度条或者等待界面提示用户等待系统响应。
- 按钮必须提供快捷方式，快捷键的取值符合其它 MIS 项目的共同习惯。

功能	快捷键	说明
增加	Ctrl+N	增加一个新的内容
明细保存	Alt + A	添加或保存一个明细资料（如一个报刊亭的报刊批销费率明细资料）
修改	Ctrl+E	修改当前内容
取消	Ctrl+C	撤消做的修改
删除	Ctrl+D	删除当前内容
明细移除	Alt +D	移除一个明细资料
保存	Ctrl+S	保存当前内容
第一条	Ctrl+Home	到第一条记录
上一条	PageUp	到上一条记录
下一条	PageDown	到下一条记录
最后一条	Ctrl+ End	到最一条记录
打印	Ctrl+P	打印当前内容
退出	Ctrl+X	退出当前 Form
换页	Ctrl+Tab	页面间的切换（控件数较多时采用多页面显示的情况）

4. 美观与协调性原则

- 图形按钮的图标必须具有代表性能产生感官上的功能认同。
- 长宽的比较接近黄金分割定律。
- 布局要合理切忌过于密集或过于空旷，合理利用空间。
- 字体统一采用宋体 5 号字大小，所有控件排列要力求整齐，左右都能对齐，Label 两个字的中间空 4 个空格，三个字的中间空两个空格。
- 前景色与背景色搭配要合理，不宜反差太大，要考虑色盲用户的使用，且尽量能融合到 HTML 页面的基本色调中去。
- 所有按钮大小基本相近，功能相近的按钮放在一起，退出按钮一般放在右下角，与最右边的控件右边对齐。
- 类似结构的 ActiveX 控件界面的整体布局保持一致。
- 所有 ActiveX 控件的界面风格要保持一致，字体、颜色要相同。

五、 组件开发注意事项

略

六、 开发制度

为了保证开发工作的顺利进行，规范控制代码的并行开发机制，特做如下规定：

- 开发人员必须按照时间的规定严格完成程序任务，每周五上午可以公布下周的详细任务清单。
- 测试人员公布的 BUG 由程序的初始编写人员负责修正。
- 修改通用模块（工具类等）、控件、组件要通知相关程序人员做适应性调整。
- 所有开发文档及软件源代码全部存放在服务器上，用 Visual SourceSafe 统一管理，程序员改动程序之前必须 check out,每日改动后要 check in。
- 系统所有原程序及其它资料每周进行备分，目录用日期表示。
- 存储的目录结构如下：

```
\\...\NewsPaper
        Server(服务器端构件)
         ── Source
              ── CData    数据层组件
                   ..........
              DLL
        Client(客户端构件)
              Source
                   CData    数据层组件
                    ..........
              OCX
        HomePage

        Graphic（图片 bmp、icon）
              BMP
              ICON
        Document
```

- 模块命名方式

DLL ——》采用 NP（NewsPaper 简称） + 模块英文大小写简称，如：NPCustMangage

CLS ——》采用模块英文大小写简称的前两位 + CLASS 的英文大小写简称,如：ClsCustMain

客户端构件命名方式

OCX ——》 Onp +模块英文大小写简称

- 网页命名方式

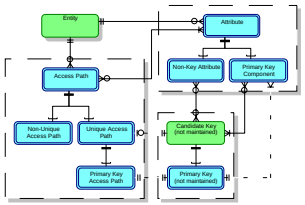
必须采用直观的、顾名思义的名称，主页用 Index.htm

- 开发人员必须每周登记工作日志。
- 公共函数和方法公开制度

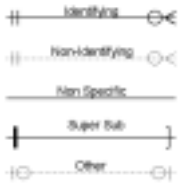
系统将提供两个工具组件 Utility.Dll 和 Public.Dll，其中前者是用于当前系统的，后者不但可以用于当前系统，而且可以用于公司的其它项目。程序开发人员在开发过程中用到某些方法，如果觉得可以归纳到以上两个组件中去，则可以向苏康胜提出。验证后加入组件。

Access Paths

Access Paths specify the attributes that can be used to locate an individual record that is represented by an entity. Access paths correspond to indexes in the physical data model.



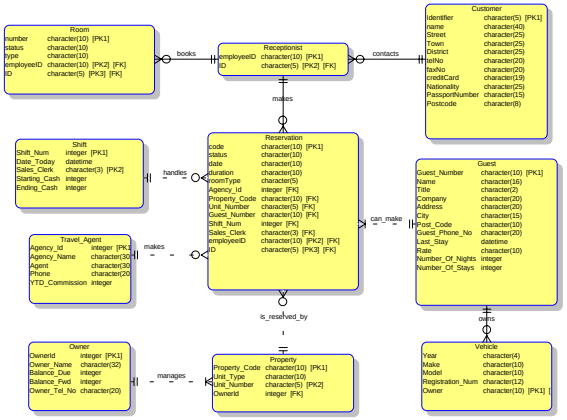
Relation Lines



Relation lines 表示实体间的关系，可能是父 / 子关系（parent/child relationship）、 Super-sub types 或 non-specific。 The nature of the parent/child relationship can be further categorised as Identifying or Non-Identifying.

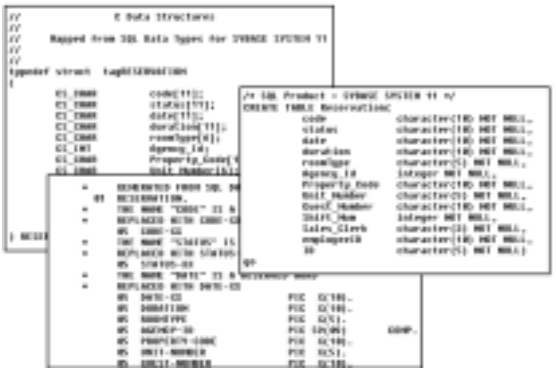
Physical Data Models

实体资料模型（**PDM, Physical Data Model**） 是用来描述一个特定的资料库，因此，实体资料模型不同于实体关系图，它只建构一个已完成的资料库模型。实体资料模型（**PDM**）可以直接转换成实体关系图（**ER Diagram**），相同地，实体关系图 **ER Diagram**）也可以直接转换成实体资料模型（**PDM**）。



Schema Generation

SA Schema Generator 准予您由实体资料模型（PDM）产生资料库架构，并且支援最普遍使用的资料库管理系统(DBMS)，SA Schema Generator 能输入至一个档案，或这个资料库能经由现存的 ODBC 做修改。

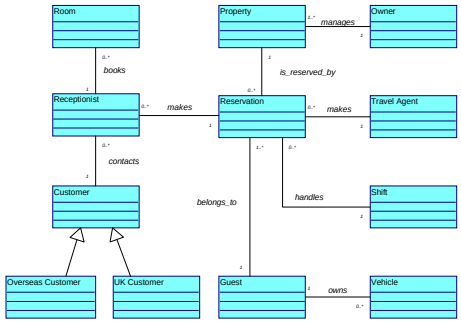


Reverse Data Engineer

资料逆向工程（Reverse Data Engineering），能让您将已存在的资料库结构转换成实体资料模型**(PDM)**。转换介面也提供第四代程式语言(4GL)应用程式，例如：Visual Basic, PowerBuilder, Delphi and Magic。

ER Diagram to OO Model

您能由实体关系图**(ER Diagram)**自动产生一个 UML Class diagram ;实体(Entities)对映到类别(Classes)，属性(Attributes)对映到类别属性(Class Attributes)。



System Architect 2001

Data Modelling

Quick Reference

POPKIN SOFTWARE

戊丰科技股份有限公司
台北市敦化南路一段 21 号 4 楼之二
Tel：(02)2570-1877 / Fax：(02)2570-1876

Methodology Outline

System Architect 2001 提供对商业模型、物件及元件模型、关联性资料模型及结构性分析及设计等四个面向的整合性支援。

Data modeling 包含实体关系模型（ Entity Relation models with subject-areas）、独立的实体模型（physical models）、 schema generation 以及资料逆向工程（reverse data engineering）。

System Architect 2001 provides the capability to build multiple project data models in a project encyclopedia. Project Data Model 能由实体关系模型图(Entity Relation Model Diagram) 及一个或多个 Subject Area Diagrams 所构成。图型及关系的定义显示出一个资料库系统，它可能是一个单一的资料库或一个分散式资料库系统。

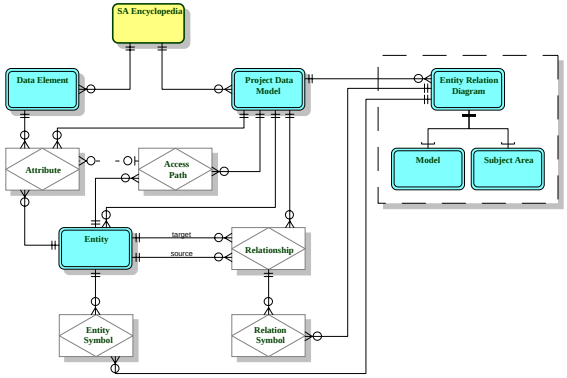
The Model Diagram 包含 project data model 内所有的实体及关系。The Model Diagram is built upon subordinate Subject Area diagrams. 每一个 Subject Area diagram 表示 Model Diagram 的一部份。

System Architect 2001 Features

- 32 位元使用者界面，及完整的企业模型整合工具。
- 多合一的电脑辅助设计工具（CASE tool ），包含商业描述、物件导向、元件设计及资料模型。
- 支援的法论包括： UML, IDEF0, IDEF3, IDEF1X, Yourdon, Ward-Mellor, Gane & Sarson, SSADM, Catalyst, OMT, Booch and Shlaer/Mellor
- 支援多人使用的网路版本，整合 Microsoft VBA、互动式关联矩阵(Matrix)及影响分析。
- 可以产生 Microsoft Word 文件及 HTML 网页文件。
- C++, Java, Dynasty & Magic – Generation & Reversing, HTML Generation, Simulation.

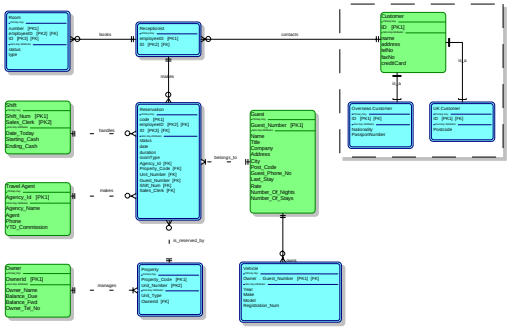
Project Data Model

Project Data Model 能由实体关系模型图(Entity Relation Model Diagram) 及一个或多个 Subject Area Diagrams 所构成。



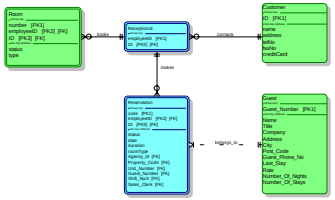
Model Diagram

The Model Diagram is built upon subordinate Subject Area diagrams. Subject Area diagram 的改变，会自动反映在 Model Diagram。



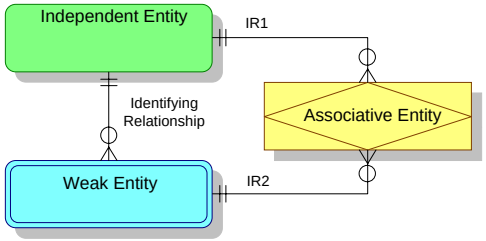
Subject Area Diagram

每一个 Subject Area diagram 表示 Model Diagram 的一部份。



Entity Symbols

实体（**Entity**）表示您企业组织内用来记录的资料内容，包含人、事、物、概念等实体。System Architect 2001 提供所有的实体符号(Entity Symbols)。



System Architect 2001 自动改变实体符号，以反映出适当的从属关系。

Entity Attributes

属性（**Attribute**）是定义实体（**Entity**）与资料项目（**Data Item**）间的关系。

The Attributes grid 准许您定义目前的实体及资料项目属性，也能被重复使用在其他的 SA design activities, process modelling, Data Flow diagrams.