X-Programmer 总第2期
软件工程的播种机
www.umlchina.com
非程序员 2001 6

目录

方法

用 UML 设计 Java 应用程序……..1
分析模式学习笔记之二：责任模式……...20
分析模式学习笔记之三：观察和测量……...26
用户需要什么-软件的工程可用性（一）……..38

过程

深刻理解 CMM－成功的关键……41
项目管理规范-RUP 管理实施（一）……44
成功项目管理的秘密……..59

工具

UML 相关产品价格……63

论坛

用 usecase 驱动 ooa,ood,oop 的困惑……..70
如何确定 actor……..71
类之间设置成"关联"or"依赖"
似乎全在个人喜好……..71
Rose 等其他 Case 工具使用的疑问……...73
中国软件业什么也不缺，就缺“笨蛋”……74

联系方法

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

播种机：http://www.umlchina.com

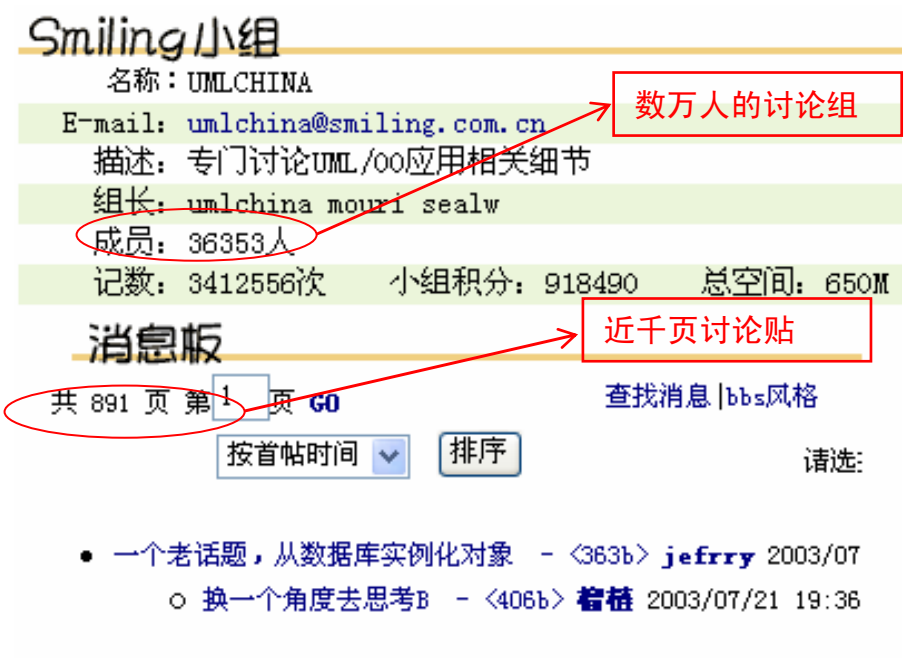
征稿

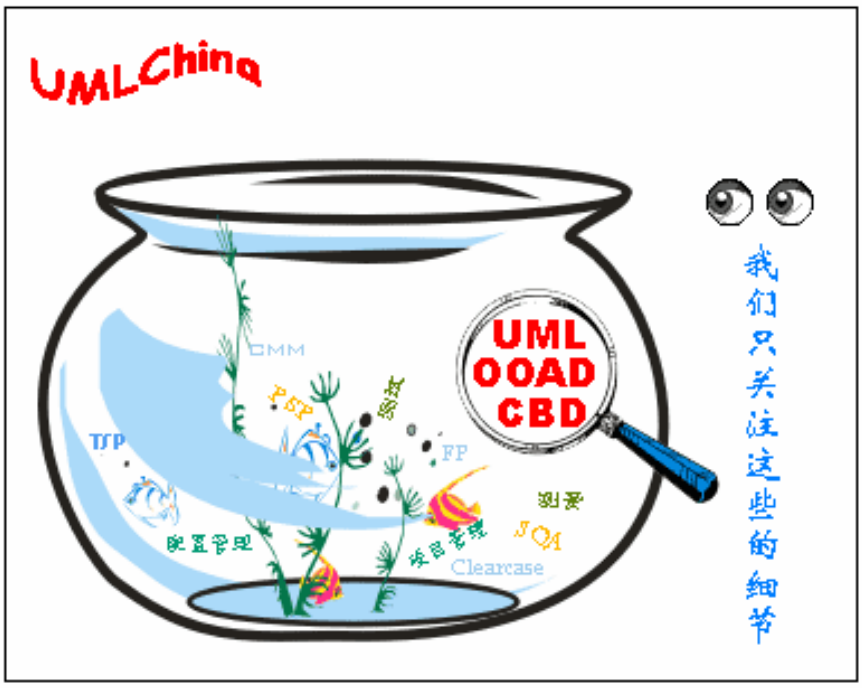
关于需求，设计，构造，测试，维护，配置管理，管理，过程，工具，质量... 原创或翻译均可。

请点击查看详情>>

征稿：
Using Patterns to Integrate UML Views
Interface Hall of Shame
Using the Software CMM in Small Organizations
Looking Back, Looking Ahead

请点击查看详情>>





UMLChina提供以下服务：团队内训，项目指导

用 UML 设计 Java 应用程序

讨论

精通用统一建模语言实现 Java 的面向对象技术

Hans-Erik Erikkson, Magnus Penker 著，刘忠(caulzhong@sina.com) 译

本文的案例学习提供了一个例子，说明如何将 UML 用在现实中。一个处理图书馆借阅和预定图书和杂志的应用程序，可以大到足够检验 UML 解决现实问题能力的程度。但是如果太大的话，则不适合在杂志上发表。

在分析模型中，用用例和域分析描述了应用程序。我们进一步把它扩展成设计模型。在设计模型中，我们描述了典型的技术解决方案细节。最后，我们编写了一段 Java 代码（代码连同完整的分析和设计模型放在网上，以一种包括评估版在内的 Rational Rose 能够识别的格式在线提供。）

必须注意，这里只是一个可行的解决方案。可能会有许多其他的解决方案。没有绝对正确的方案。当然，有的方案更好一些，但只有不断的实践和努力的工作才能掌握相应的技能。

1. 需求（Requirements）

典型地，由系统最终用户的代表写出文本形式的需求规范文档。对于该图书馆应用程序来说，需求规范文档应该类似于这样：

1. 这是一个图书馆支持系统；
2. 图书馆将图书和杂志借给借书者。借书者已经预先注册，图书和杂志也预先注册；
3. 图书馆负责新书的购买。每一本图书都购进多本书。当旧书超期或破旧不堪时，从图书馆中去掉。
4. 图书管理员是图书馆的员工。他们的工作就是和读者打交道并在软件系统的支持下工作。
5. 借阅人可以预定当前没有的图书和杂志。这样，当他所预定的图书和杂志归还回来或购进时，就通知预定人。当预定了某书的借书者借阅了该书后，预定就取消。或者通过显式的取消过程强行取消预定。
6. 图书馆能够容易地建立、修改和删除标题、借书者、借阅信息和预定信息。
7. 系统能够运行在所有流行的技术环境中，包括 Unix, Windows 和 OS/2，并应有一个现代的图形用户界面 (GUI)。
8. 系统容易扩展新功能。

系统的第一版不必考虑预定的图书到达后通知预定人的功能，也不必检查借书过期的情况。

2. 分析（Analysis）

系统分析的目的是捕获和描述所有的系统需求，并且建立一个模型来定义系统中主要的域类。通过系统分析达到开发者和需求者的理解和沟通。因此，分析一般都是分析员和用户协作的产物。

在这个阶段，程序开发者不应该考虑代码或程序的问题；它只是理解需求和实现系统的第一步。

2．1 需求分析（Requirements Analysis）

分析的第一步是确定系统能够做什么？谁来使用这个系统？这些分别叫角色（actors）和用例（use cases）。用例描述了系统提供什么样的功能。通过阅读和分析文档，以及和潜在的用户讨论系统来分析用例。

图书馆的角色定为图书管理员和借书人。图书管理员是软件系统的用户；而借书者则是来借阅或预定图书杂志的客户。偶尔，图书管理员或图书馆的其他工作人员也可能是一个借书者。借书者不直接和系统交互，借书人的功能由图书管理员代为执行。

图书馆系统中的用例有：

1. 借书
2. 还书
3. 预定
4. 取消预定
5. 增加标题
6. 修改或删除标题
7. 增加书目
8. 删除书目
9. 增加借书者
10. 修改或删除借书者

由于一本书通常有多个备份，因此系统必须将书的标题和书目的概念区分开。

图书馆系统分析的结果写在 UML 用例图中，如图 1 所示。每一个用例都附带有文本文档，描述用例和客户交互的细节。文本是通过与客户讨论得到的。用例“借书”描述如下：

1．如果借阅者没有预定：

- 确定标题
- 确定该标题下有效的书目
- 确定借书者
- 图书馆将书借出
- 登记一个新的借阅

2．如果借阅者有预定：

- 确定借书人
- 确定标题
- 确定该标题下有效的书目
- 图书馆将相应的书目借出
- 登记一个新的借阅
- 取消预定

除了定义系统的功能需求之外，在分析过程中用例用于检查是否有相应的域类已经被定义，然后他们可以被用在设计阶段，确保解决方案可以有效地处理系统功能。可以在顺序图中可视化实现细节。

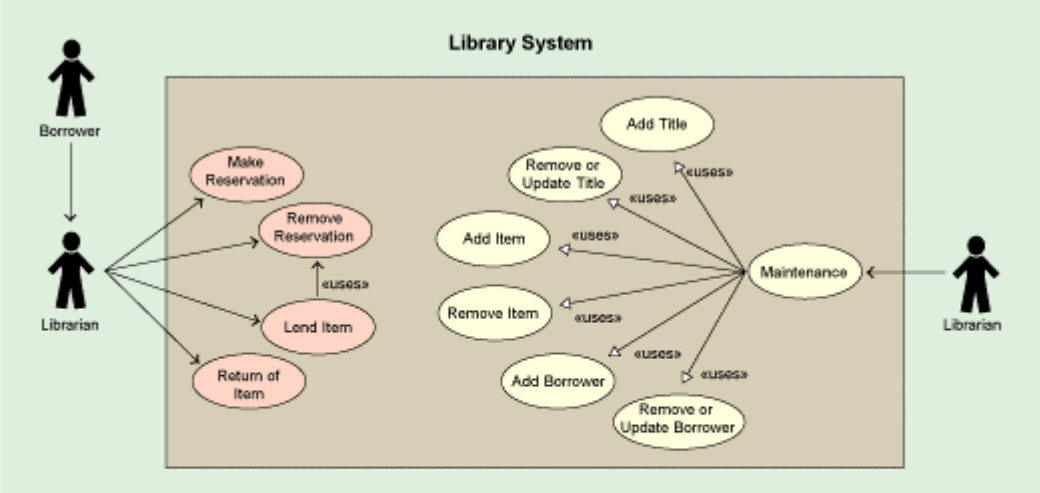


Figure 1. Actors and Use Cases. The first step in analysis is to figure out what the system will be used for and who will be using it. These are the use cases and actors, respectively. All use cases must begin with an actor, and some will also end with an actor. Actors are people or other systems that are outside of the system you are working on. A printer or a database might be an example of an actor. This system has two actors, borrowers and the librarian. Each of the use cases will be fleshed out in text defined via discussions with the user/customer.

图 1：角色和用例。分析中的第一步就是指出系统能被用来做什么，谁将去使用它。它们分别就是用例和角色。所有的用例必须始于角色，而且有些用例也结束于角色。角色是位于你所工作的系统外部的人或其他系统。一台打印机或一个数据库都可能是一个角色。本系统有两个角色：借阅者和图书管理员。通过与用户或客户的讨论，可以将每一个用例用文字进行说明。

2．2 域分析（Domain Analysis）

系统分析也详细地列出了域（系统中的关键类）。为了导出一个域分析，可以阅读规范文档（specifications）和用例，查找哪一些概念应该被系统处理。或者组织一个集体讨论，在用户及领域专家共同的参与下指出系统中必须处理的关键概念，以及它们之间的关系。

图书馆系统中的域类如下：borrowerinformation（如此命名是为了与用例图中的角色 borrower 区分开来），title，book title, magazine title, item, reservation 和 loan。这些类以及它们之间的关系记录在类图文档中，如图 2 所示。域类定义为 Business object 版型，Business object 版型是一个用户自定义的版型，指定该类的对象是关键域的一部分，并且应该在系统中持久存储。

其中有些类有 UML 状态图，用来显示这些类的对象可能具有的不同状态，以及触发他们的状态发生改变的事件。该例子中有状态图的类是 item 和 title 类。

用例 lend item（借阅者没有预定的情况）的顺序图显示在图 3 中。所有用例的顺序图都可从在线模型中查到。

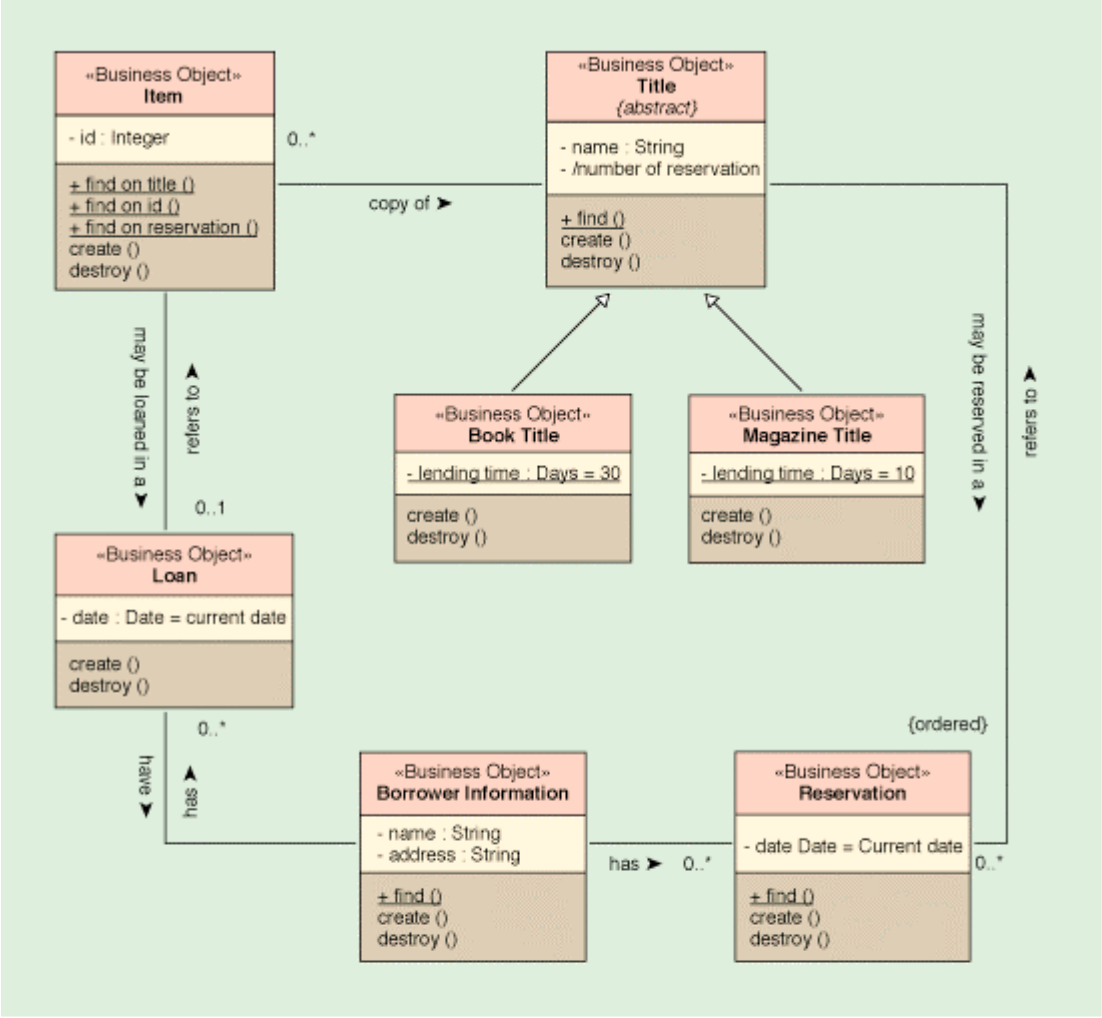


Figure 2. Domain Class Structure. Domain analysis itemizes the key classes in the system. For each object that calls a method on another object, there is a line joining the classes, to show that relationship. Each class rectangle is divided into three compartments. The top compartment contains the name of the class, the middle the attributes of the class, and the bottom the methods of the class. The lines between classes are associations, which indicate an object in one has called a method in the other. If you look closely, you will see "0..1" near the Loan end of the association between Loan and Item, which represents the multiplicity of the association. The multiplicity of "0..1" means that Item knows about between zero and one Loans. Other possible multiplicities are "0..*", meaning zero or more, "1", meaning exactly one, "0", meaning exactly zero, and "1..*", meaning one or more.

图 2：域类结构。域分析详细说明了系统中的关键类。对每一个对象而言，如果它调用了其他对象的方法，那么在他们之间就用一条直线连结起来，以显示他们之间的关系。每一个代表类的四边形被分成了三部分，最顶层包括类的名称，中间一层是类的属性，最底层是类的方法。类之间的直线是关联，用来指出一个对象调用另一个对象的方法。如果再仔细看，将会发现在 Loan 和 Item 之间的关联关系中靠近 Loan 的一端有“ 0..1 ”,这代表关联的重数。重数“ 0..1 表示 Item 可以感知 0 个到 1 个 loan。其他可能出现的重数还有：“ 0..* ”表示 0 或多；“ 1 ”表示就是 1；“ 0 ”表示就是 0，“ 1..* ”表示 1 或多。

当对顺序图建模时，必须提供窗体和对话框作为人机交互的界面。在本分析当中，只要知道借书、预定和还书需要窗体就可以了。在此，详细的界面不必考虑。

为了把系统中的窗体类和域类分开，所有的窗体类组织在一起放在 GUI Package 包中。域类组织在一起放在 Business Package 包中。

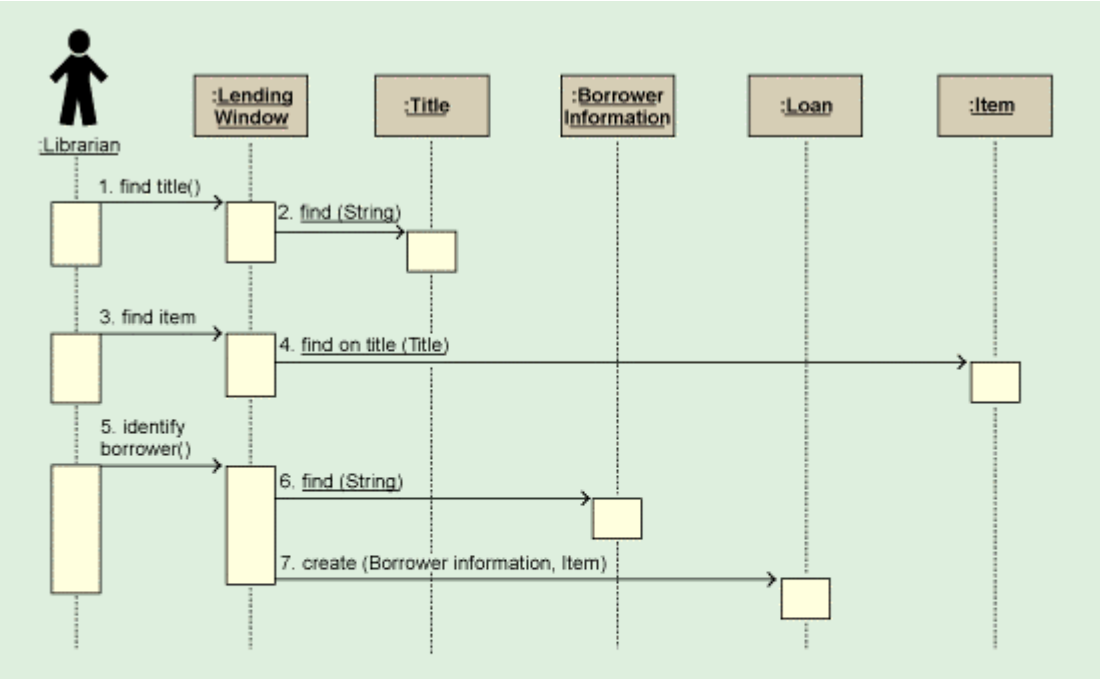


Figure 3. Sequence Diagram for the Lend Item Scenario. Scenarios are particular paths through a use case. A scenario always starts with an actor, who is outside the system. It then traces a complete path through the system until the action is finished from all actors' points of view. The UML notation used to diagram a scenario is a sequence diagram. This sequence diagram shows the case for Lending when the borrower does not have a reservation for the title. Across the top are the objects that are interacting. Time goes down the page. So, first the Librarian tries to find a title. The object labeled "Lending Window" is the user interface, treated during analysis as a single coarse-grained object. Each arrow across in the sequence diagram is a call of a method of the object at the head end of the arrow, by the object at the tail end of the arrow.

图 3：Lend item 场景的顺序图。场景是从头到尾实现一个用例的一次特定的过程。场景总是始于角色，而角色是属于系统外部的。场景描绘了从所有角色的观点出发，完成一次系统动作的完整过程。UML 在用顺序图来图示场景。本用例图显示了在借阅者没有预定图书的情况下的 Lend 用例。横在图的顶部的是参与交互的对象。自上而下表示时间的流逝。首先，图书管理员尝试去查找标题。标有“Lending Window”的是用户界面，在分析阶段作为一个粗略的对象。横在顺序图中的每一个箭头都是一次方法的调用，箭头的首端是调用的对象，箭头的末端是被调用的对象。

3. 设计（Design）

设计阶段对分析模型进行扩展并将模型进一步细化，并考虑技术细节和限制条件。设计的目的是指定一个可行的解决方案，以便能很容易地转变成为编程代码。

设计可以分成两个阶段：

体系结构设计阶段（Architecture Design）。这是一个从较高层次的进行的设计，用来定义包（子系统），描述包之间的依赖性及通信机制。很自然，目的是要设计一个清晰简单的体系结构，有很少的依赖性，而且尽可能避免双向依赖。**详细设计阶段（Detailed Design）。**在此阶段，所有的类都详尽地进行描述，给编写代码的程序员一个清晰的规范说明。UML 中的动态模型用来说明类的对象如何在特定的情况下做出相应的表现。

3．1 体系结构设计

一个良好的体系结构设计是一个可扩展的和可改变的系统的基础。包可能关注特定的功能领域或关注特定的技术领域。把应用程序逻辑（域类）和技术逻辑分开是至关重要的，这样不管哪一部分的改变都不会影响其他的部分。

本案例的包或叫子系统如下：

User-Interface Package 包。该包中的类基于 Java AWT 包，java AWT 是一个用来书写用户界面应用程序的 Java 的标准库。该包和 Business-objects Package 包协作。Business-objects Package 包包含那些实际存储数据的类。UI 包调用 Business 对象的操作，对他们进行取出或插入数据操作。

Business-object Package。该包包括域类，这些域类（如 borrowerinformation，title，item，loan 等）来自于分析模型。设计阶段完整地定义了这些类的操作，并增加了一些其他细节来支持持续存储。Business-object 包与 Database Package 进行协作。所有的 Business-object 类必须继承 Database Package 中的 persistent 类。

Database Package。Database Package 向 Business-object 包中的类提供服务，以便他们能够持续地存储。在当前版本中，persistent 类将把它的子类的对象存储到文件系统的文件中。

Utility Package。Utility Package 包含了一些服务，用来被系统中其他包调用。当前，ObjId 类是该包中的唯一的类。用来被整个系统包括 User-Interface，Business-Object 和 Database package 使用。

这些包的内部设计如图 4 所示.

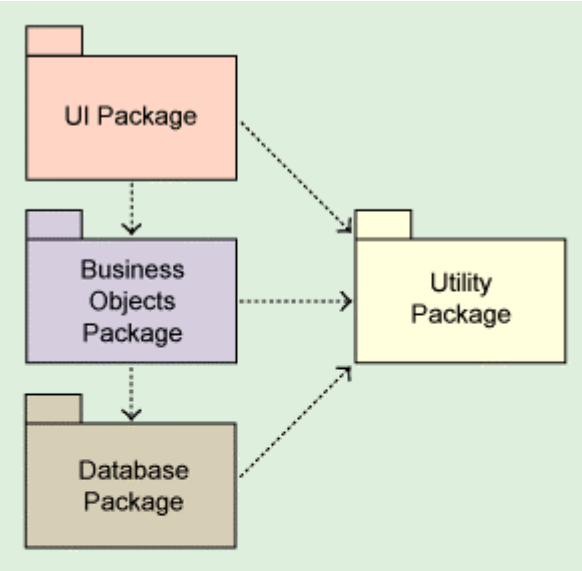


Figure 4. Architectural Overview of Library Application. This class diagram shows the application packages and their dependencies. The Database package provides persistence. The utility package provides the Object ID class. The Business-Objects package contains the domain classes detailed in Figure 5. Finally, the UI package, based in this case on the standard Java AWT library, calls operations on the business objects to retrieve them and insert data into them.

图 4：图书馆应用程序体系结构设计总览。本类图显示了应用程序包以及它们之间的依赖性。Database 包提供了 persistence 类。Utility 包提供了 Object ID 类。Business-Object 包包含了域类（详细情况参见图 5）最后，UI 包（在本例中它是基于标准 Jaa AWT 库）调用 business 对象中的操作来实现对他们的数据存取操作。,

3．2 详细设计

细节设计描述了新的技术性的类，如 User-Interface 和 Database 包中的类，并且丰富了分析阶段所形成的 Business-Object 类。类图、状态图和动态图用的还是分析阶段所形成的图，但对他们定义的更加详细，具有了更高的技术水平。在分析阶段对用例进行的文字性描述在此用来证明用例在设计阶段也能被处理。顺序图就是用来说明用例如何在系统中被实现的。

Database Package。应用程序必须有持续存储的对象。因此，必须增加数据层来提供这样的服务。为简单起见，我们将对象以文件的形式保存在磁盘上。存储的细节被应用程序隐藏起来，只需调用诸如 store（），update（），delete（）和 find（）这样的公共操作即可。这些都是 persistent 类的一部分，所有需要

持续对象的类必须继承它。

对类进行持续处理的一个重要因子就是 ObjId 类。它的对象用来引用系统中的任何持续对象（不管这个对象是在磁盘上还是已经被读进了应用程序之中）。ObjId 是 Object Identity 的简写，它是一个广为应用的技术，用来有效地处理应用程序中的对象引用。通过使用 object identifiers，一个对象 ID 能被传递到普通的 persistent.getobject（）操作中，进而该对象将被从持续的存储体中取出或存储。通常情况下，这些都是通过每个持续类的一个 getobject 操作完成的。当然，持续类同时也作一些检查或格式转换的操作。一个对象标识符也能作为一个参数很容易地在两个操作之间传递（例如，一个查找特定对象的查询窗口可以将它的查询结果通过 object id 传递给另一个窗口 ）。

ObjId 是一个系统中所有的包（User Interface , Business Object 和 Database）都能使用的通用类，所以在设计阶段它被放置在 Utility 包中，而不是放在 Database 包中。

当前对 persistent 类的实现还能改进。为此，定义 persistent 类的接口，方便持续存储的改变。一些备选的方案可能是：将对象存储在一个关系数据库中或存储在面向对象的数据库中，或使用 Java 1.1 所支持的持续对象来存储他们。

Business-Object Package。设计阶段的 Business-Object 包基于相应的分析阶段的放置域类的包。类和类之间的关系以及他们的行为继续保留，只是被描述的更为详细，包括他们的关系和行为如何被实现。

分析模型中的一些操作中被翻译成设计模型的操作，另一些改了名字。这是很正常的事，因为分析阶段得到的是每一个类的草图，而设计阶段是对系统的详细描述。因此，设计模型的操作必须有设计良好的特征值和返回值（由于空间限制，图 5 没有显示，但他们在在线模型中都有）。注意以下所列的设计和分析阶段的变化：

1. 系统的当前版本不要求检查书目是否按时归还，也不要求处理预定的次序。因此没有在 loan 和 reservation 类中设置日期属性。
2. 除了最长借阅期外，对杂志和书标题的处理方式是一样的。因此分析阶段的子类 magazine title 和 book title 被认为在设计阶段是不必要的，而是在 title 类中增加 type 属性来指出该标题引用的是一本书还是一本杂志。在面向对象的设计中不存在设计不能简化分析的说法。

如果认为有必要的话，在将来的版本中这些简化都可以很容易地被取消。

分析阶段的状态图也在设计阶段细化了。状态图显示了如何表示状态及如何在系统中处理状态。设计阶段的 title 类的状态图如图 6 所示。其他的对象可以通过调用如图所示的操作 addreservation（）和 removereservation（）来改变 title 对象的状态。

User-Interface Package。User-Interface Package 位于其他包的“上面”。在系统中它为用户提供输出信息和服务。正如上面曾经提到的，该包是基于标准 Java AWT（abstract windows toolkit）类的。

设计模型中的动态模型放置在 GUI 包中，因为所有和用户的交互都从用户界面开始。在此声明，顺序图用来显示动态模型。用例在设计模型中的实现通过顺序图被详细地显示出来，包括类中的实际操作。

顺序图由一系列的交互构成。在实现阶段（编码），考虑到具体情况，可能会有更多的交互。图 7 显示了 add title 用例的顺序图。实际的操作和特征值从在线模型代码中可以看到。

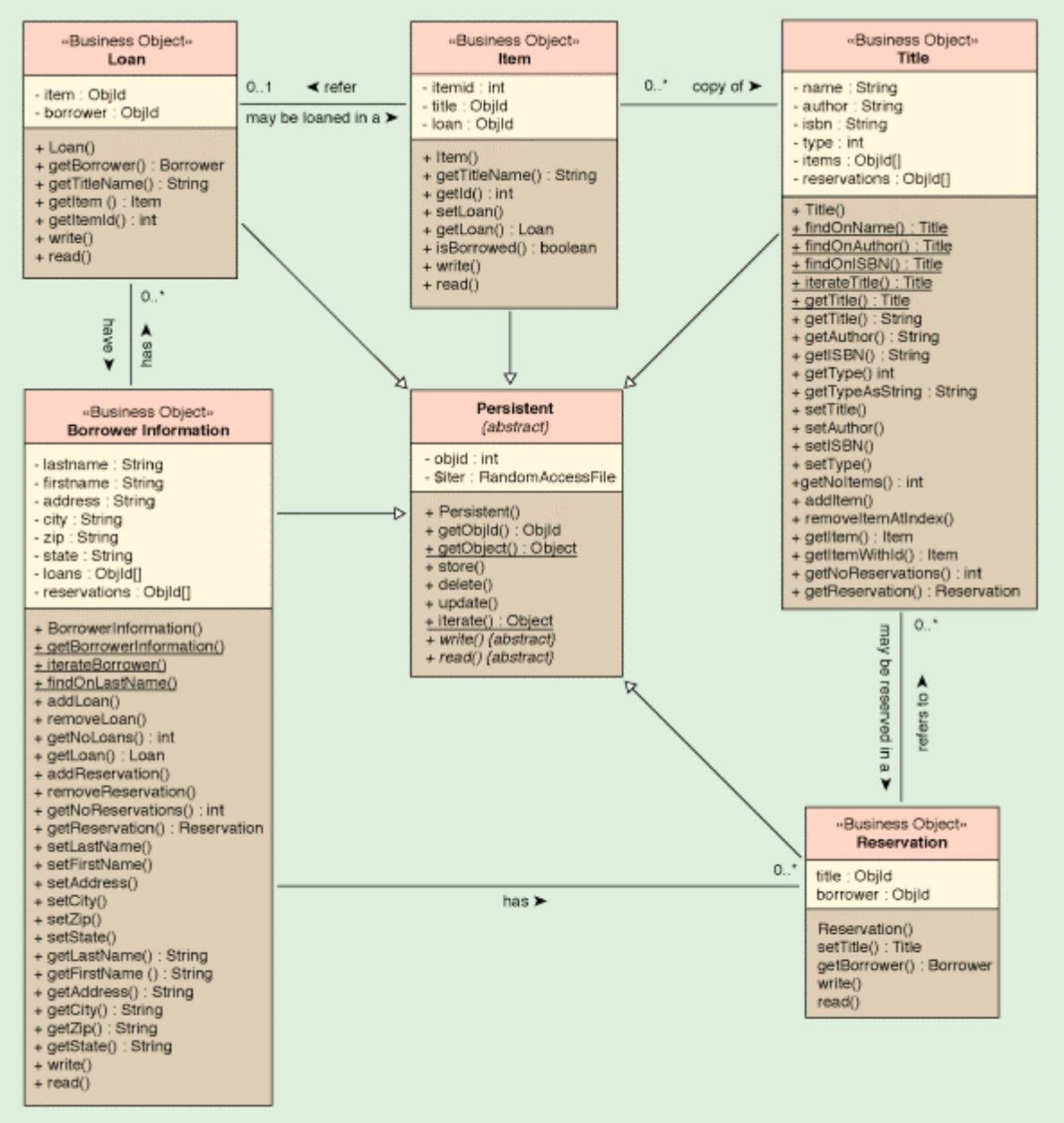


Figure 5. Business-Objects Design. This diagram fleshes out the design of the various classes in the Business-Objects package. Design is when the details of the model are settled. Interfaces are more fully specified, data types for attributes are chosen, and so on.

图 5：商业对象设计（Business-Object design）。本图描述了在 Business-Object 包中的不同类的设计。设计包括定型模型，更完全地定制界面，给属性选择数据类型等等。

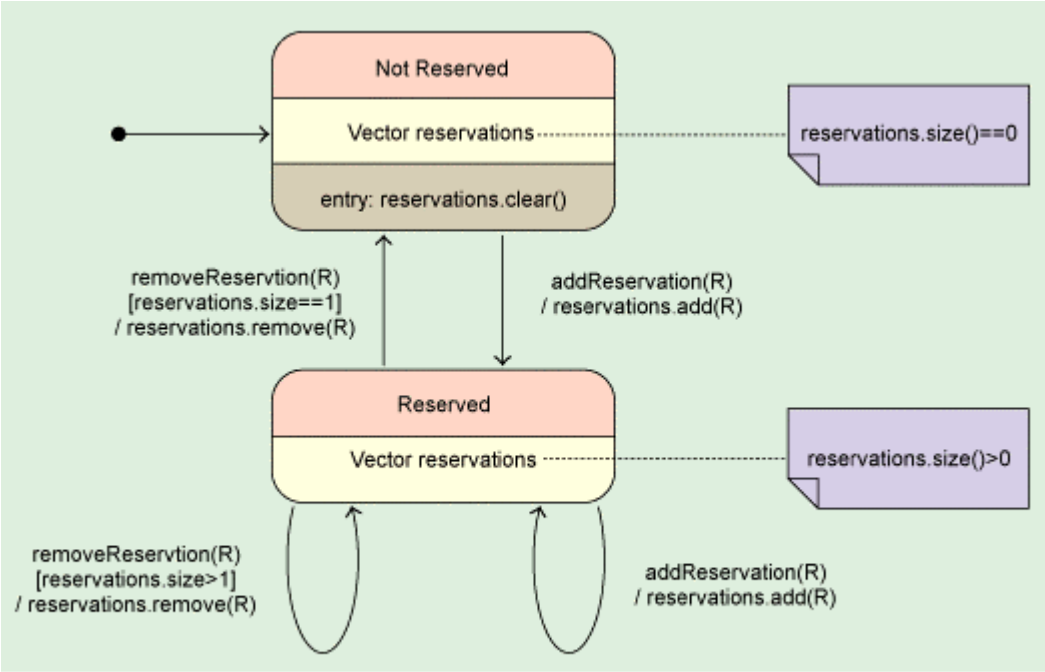


Figure 6. State Diagram for Title. The states for Title, reserved and unreserved, are implemented in the design by using a Vector called "reservations."

图 6：Title 的状态图。Title 具有预定和非预定状态，在设计中，通过称为“reservations”的矢量来实现。

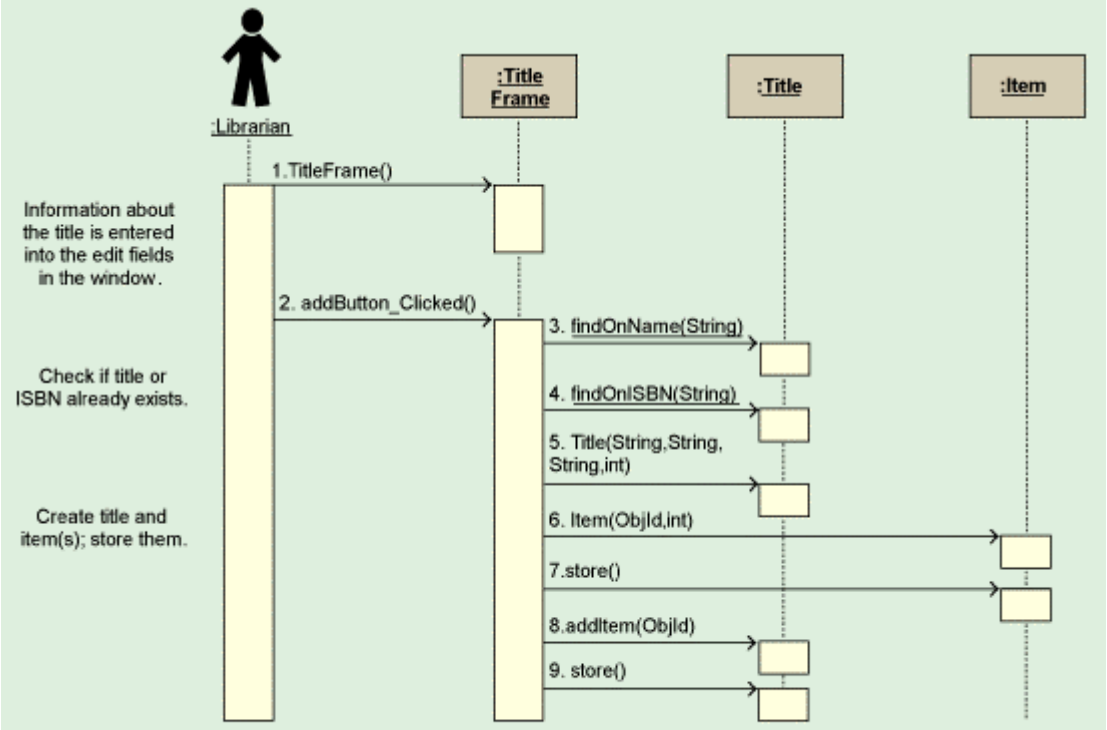


Figure 7. A Sequence Diagram for Add Title. The details of the UI issues addressed in this figure are beyond the scope of this article.

图 7：Add Title 的顺序图。本图中所涉及到的用户界面问题的详细情况已经超出了本文的讨论范围。

协作图可以作为顺序图的替代，如图 8 所示：

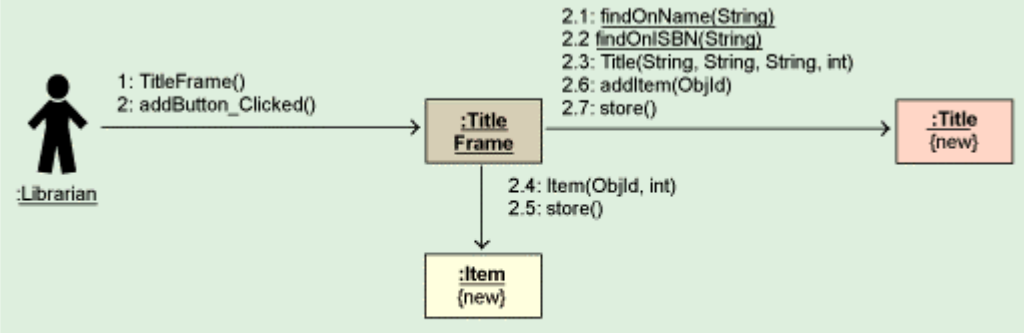


Figure 8. A Collaboration Diagram for Add Title. The details of the UI issues addressed in this figure are beyond the scope of this article.

图 8：Add Title 的协作图。本图中涉及到的用户界面问题的详细情况已经超出了本文讨论的范围

3．3 用户界面 设计（User-Interface Design）

设计阶段的一个特定的活动是创建用户界面。

图书馆系统的用户界面基于用例，分为以下几部分，每一部分都在主窗体菜单上给出一个单独的菜单项。

Functions：实现系统基本功能的窗体，通过它可以实现借阅、归还和对图书的预定。

Information：查看系统信息的窗体，收集了借阅者和图书的信息。

Maintenance：维护系统的窗体，添加、修改和删除标题、借阅者和书目。

图 9 显示了一个 User-Interface Package 中类图的例子。其中包含了典型的 AWT 事件句柄。按钮（button）、标签（label）和编辑（edit）等的属性没有显示。

典型地，每一个窗体都给出了系统的一个服务，并且映射一个最初的用例（尽管并非所有的用户界面都必须从用例中映射）。创建成功的用户界面已经超出了本文所讨论的范围。在此邀请读者来考虑用 symantec visual cafe 环境开发的本应用程序的 UI 代码（已经放在网上）。

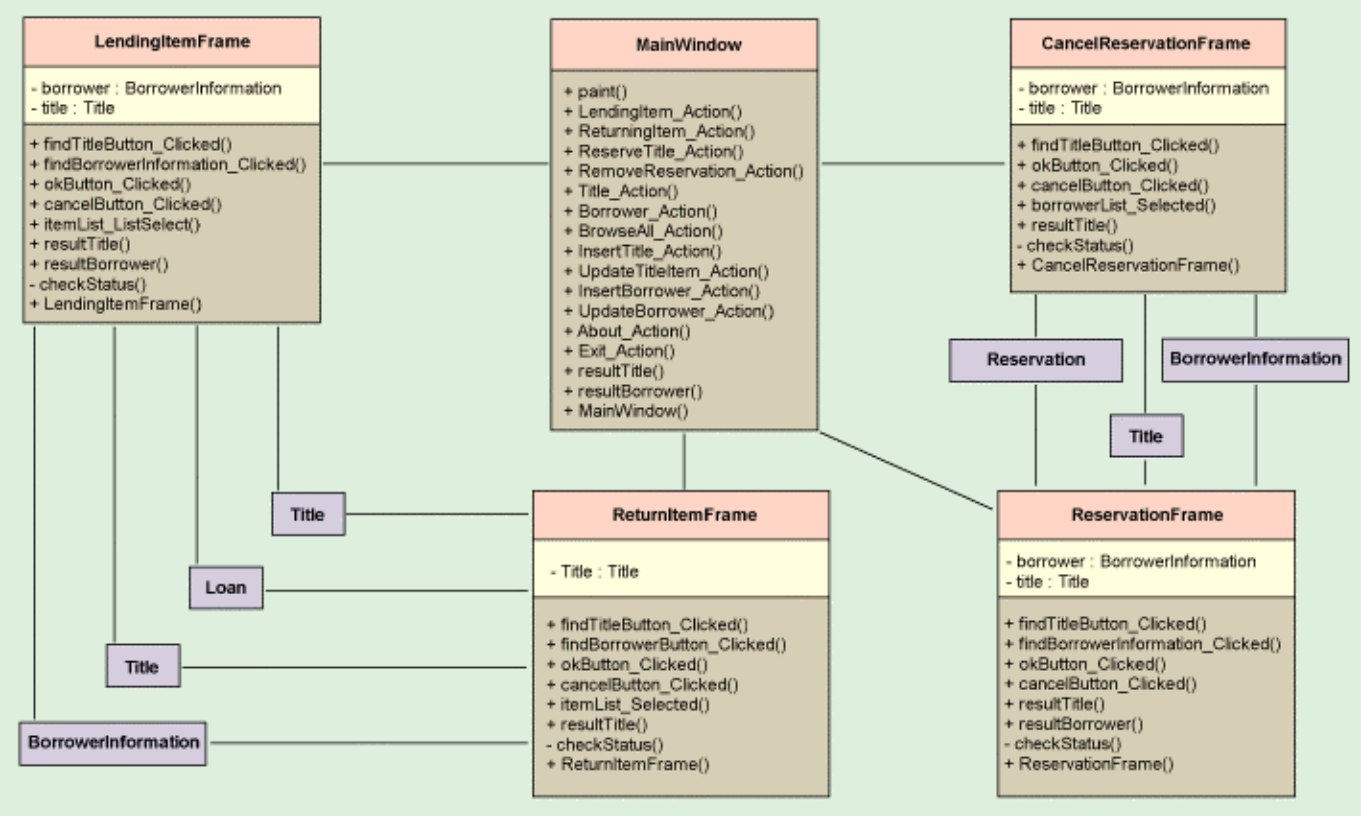


Figure 9. Functions Class Diagram Model. User-interface classes in the Functions Menu are typical in having all one-to-one associations, implying that the associated window class at some point is created or that the associated business object class is accessed.

图 9：功能类图模型。功能菜单中的用户界面类一般都有 1 对 1 的关联关系，表示需要建立关联的窗口类，或者需要访问关联的商业对象类。

4．实现（Implementation）

在构造或称实现阶段进行程序编写。该应用程序要求能运行在几个不同的处理器和不同的操作系统上，因此选择 Java 来实现系统。Java 可以轻松地将逻辑类映射为代码组件，因为在类和 Java 代码文件之间有 1 对 1 的映射。

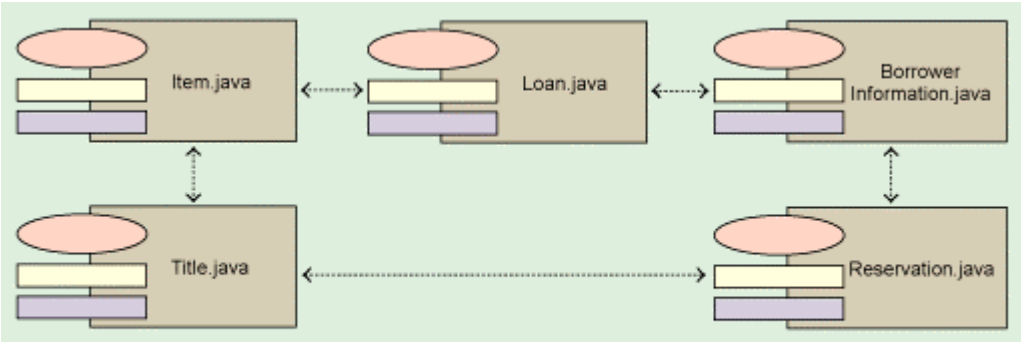


Figure 10. Component Diagram Showing Dependencies. The associations of these source-code components, which implement the domain class, show bidirectional dependencies.

图 10：组件图显示了依赖性。源代码组件实现了域类，他们之间的关联显示为双向依赖性

图 10 显示，设计模型的组件视图简单地将逻辑视图中的类映射为组件视图中的组件。每个逻辑视图

包含了一个指向逻辑视图中类的连接，因此可以很容易地在不同的视图之间导航（即便象本例只是简单地使用了文件名）。由于依赖性可以从逻辑视图的类图中得到，因此组件图中没有显示组件之间的依赖性。

编写代码时，从下面的设计模型的图中取出规范说明：

1. 类规范：每一个类的规范详细地显示了必要的属性和操作。
2. 类图：类图由类构成，显示了类的结构以及类之间的关系。
3. 状态图：类的状态图显示了类可能具有的状态以及需要处理的状态转移（以及触发转移的操作）。
4. 动态图（顺序图、协作图和活动图）：涉及到类的对象，显示了类中的特定方法如何实现，或对象之间如何使用其它类的对象进行交互。
5. 用例图和规范：当开发者需要了解更多的关于系统如何被使用的信息时（当开发者感到他或她已经迷失在一片细节中），他们显示了系统被使用的结果。

很自然，设计中的某些缺陷可以在编码阶段发现。可能需要一些新的操作或修改某些操作，这意味着开发者必须改变设计模型。这在所有的工程中都会发生。重要的是将设计模型和代码同步，以便使模型能够成为系统的最终文档。

这里给出了 loan 类和部分 titleframe 类的 Java 代码。整个应用程序的 Java 代码可以从网上查到。当学习这些代码时，注意结合 UML 模型，考虑 UML 结构是如何被转变为代码的。注意以下几点：

1. Java 包规范是与类所属的逻辑视图或组件视图中相应的包等值的代码。
2. 私有属性对应于模型中指定的属性；并且，很自然 Java 方法对应于模型中的操作。
3. ObjId 类（对象标识符）被调用来实现关联。也就是说，关联通常和类一起保存（因为 ObjId 类是持续的）。

程序清单 1 的代码示例来自于 loan 类，loan 类是一个 Business-Object 类，用来存放借阅信息。由于该类主要是信息的存放处，因此程序的实现是直接的，代码也简单。大多数的功能继承了 Database 包中的 Persistent 类。该类的唯一属性是对象标识符，将 item 和 borrowerinformation 类关联起来。并且这些关联属性也在 Write（）和 Read（）操作中被保存。

试着在 Add Title 顺序图（图 7）上下文中检验一下程序清单 2 中显示的 Addbutton_Clicked（）操作。结合顺序图阅读代码，可以看出：它就是另一个对顺序图所表达的协作关系的更为详细的描述。

所有设计模型中的顺序图的编码都包含在源代码当中（操作名和类名显示在顺序图中）

5. 测试和部署(Test and Deployment)

编码结束后，UML 的使用还没有停止。例如，可以检验用例能否在已完成的应用程序中得到很好的支持。对于系统的部署来说，利用模型和本文可以做一份得心应手的文档。

6. 总结

本学习案例的不同部分由一组人分别来设计完成。他们以做实际工程的方式努力完成该工作。尽管不同的阶段和活动似乎是独立的，而且以严格的顺序来管理，但在实际工作中仍然有很多反复。设计中的经验和教训反馈到分析模型，实现阶段所发现的新情况在设计模型中更改或更新。这就是建立面向对象系统的一般方法。

本文摘录自 UML Toolkit，New York：Wiley & Sons，1998. Hans-Erik Erikkson 是一个有名的 C++和 OO 技术方面的作者。Magnus Penker 是 Astrakan 培训副主席，Astrankan 是一个瑞典专攻 OO 建模和设计的公司。

原文链接：http://www.umlchina.com/Indepth/DesignJava.htm
模型及源码：http://www.umlchina.com/zippdf/libraryjava.zip

附：主要术语中英文对照

actor：角色
use case：用例
domain：域
domain analysis：域分析
specification：规范文档
sequence diagram：顺序图
collaboration diagram：协作图
component diagram：组件图
state diagram：状态图
dependency：依赖性
attribute：属性
method：方法
operation：操作
association：关联
multiplicity：重数
class：类
object：对象
package：包
implementation：实现
deployment：部署

附：源代码

Listing 1. Loan class. Loan is a Business-object class used for storing information about a loan. Most of the functionality is inherited from the Persistent class in the Database Package.

```
//    Loan.java: represents a loan. The loan refer to one
//    title and one borrower.
//

Package bo;
import util.ObjId;
import db.*;
import java.io.*;
import java.util.*;
```

```
public class Loan extends Persistent
{
    private ObjId item;
    private ObjId borrower;
    public Loan()
    {
    }
    public Loan(ObjId it, ObjId b)
    {
    item = it;
    borrower = b;
    }
    public BorrowerInformation getBorrower()
    {
    BorrowerInformation ret =
        (BorrowerInformation) Persistent.getObject(borrower);
    return ret;
    }
    public String getTitleName()
    {
    Item it = (Item) Persistent.getObject(item);
    return it.getTitleName();
    }
    public Item getItem()
    {
    Item it =
        (Item) Persistent.getObject(item);
    return it;
    }
    public int getItemId()
    {
    Item it = (Item) Persistent.getObject(item);
    return it.getId();
    }
    public void write(RandomAccessFile out)
    throws IOException
    {
    item.write(out);
    borrower.write(out);

    }
    public void read(RandomAccessFile in)
```

```
        throws IOException
        {
        item = new ObjId();
        item.read(in);
        borrower = new ObjId();
        borrower.read(in);
        }
    }
```

Listing 2. TitleFrame Class. This is another, more detailed description of the collaboration described by the diagram in Figure 7.

```
    //   TitleFrame.java
    //

    Package ui;

    import bo.*;
    import util.*;

    import java.awt.*;

    public class TitleFrame extends Frame {
        private Title current;

        void addButton_Clicked(Event event) {
        if (Title.findOnName(titleField.getText()) != null)
        {
            new MessageBox(
            this,"A Title with that name already exists!");
            return;
        }
        if (Title.findOnISBN(isbnField.getText()) != null)
        {
            new MessageBox(
            this,"A title with the
            same isbn/nr field already exists!");
            return;
        }
        int type = 0;
        if (bookButton.getState() == true)
            type = Title.TYPE_BOOK;
```

```
else if (magazineButton.getState() == true)
    type = Title.TYPE_MAGAZINE;
else
{
    new MessageBox(this,"Please give type of title!");
    return;
}
current =
    new Title(
    titleField.getText(),
    authorField.getText(),
    isbnField.getText(),
    type);
int itemno;
if (itemsField.getText().equals(""))
    itemno = 0;
else
    itemno = Integer.valueOf(
    itemsField.getText()).intValue();
if (itemno > 25)
{
    new MessageBox(this, "Maximum number of items is 25!");
    return;
}
for (int i = 0; i <\<> itemno; i++)
{
    Item it = new Item(current.getObjId(),i+1);
    it.store();
    current.addItem(it.getObjId());
}
current.store();
titleField.setText("");
authorField.setText("");
isbnField.setText("");
itemsField.setText("");
bookButton.setState(false);
magazineButton.setState(false);

}

void cancelButton_Clicked(Event event) {
dispose();
}
```

```
public TitleFrame() {

//{{INIT_CONTROLS
setLayout(null);
addNotify();
resize(
    insets().left + insets().right + 430,insets().top +
    insets().bottom + 229);
titleLabel = new java.awt.Label("Title Name");
titleLabel.reshape(
    insets().left + 12,insets().top + 24,84,24);
add(titleLabel);
titleField = new java.awt.TextField();
titleField.reshape(
    insets().left + 132,insets().top + 24,183,24);
add(titleField);
authorField = new java.awt.TextField();
authorField.reshape(
    insets().left + 132,insets().top + 60,183,24);
add(authorField);
isbnField = new java.awt.TextField();
isbnField.reshape(
    insets().left + 132,insets().top + 96,183,24);
add(isbnField);
label1 = new java.awt.Label("ISBN / Nr");
label1.reshape(
    insets().left + 12,insets().top + 96,84,24);
add(label1);
label2 = new java.awt.Label("Author");
label2.reshape(
    insets().left + 12,insets().top + 60,84,24);
add(label2);
addButton = new java.awt.Button("Insert");
addButton.reshape(
    insets().left + 348,insets().top + 24,60,24);
add(addButton);
cancelButton = new java.awt.Button("Close");
cancelButton.reshape(
    insets().left + 348,insets().top + 192,60,24);
add(cancelButton);
label3 = new java.awt.Label("Items available");
label3.reshape(
```

```
        insets().left + 12,insets().top + 192,108,24);
    add(label3);
    itemsField = new java.awt.TextField();
    itemsField.reshape(
        insets().left + 132,insets().top + 192,36,23);
    add(itemsField);
    Group1 = new CheckboxGroup();
    bookButton =
        new java.awt.Checkbox("Book", Group1, false);
    bookButton.reshape(
        insets().left + 132,insets().top + 132,108,24);
    add(bookButton);
    magazineButton =
        new java.awt.Checkbox("Magazine", Group1, false);
    magazineButton.reshape(
        insets().left + 132,insets().top + 156,108,24);
    add(magazineButton);
    label4 = new java.awt.Label("Type");
    label4.reshape(
        insets().left + 12,insets().top + 132,108,24);
    add(label4);
    setTitle("Insert Title Window");
    //}}
    bookButton.setState(true);
    titleField.requestFocus();

    //{{INIT_MENUS
    //}}
    }

    public TitleFrame(String title) {
    this();
    setTitle(title);
    }

    public synchronized void show() {
    move(50, 50);
    super.show();
    }

    public boolean handleEvent(Event event) {
    if (event.id == Event.WINDOW_DESTROY) {
        dispose();
```

```
            return true;
        }
        if (event.target == addButton && event.id ==
            Event.ACTION_EVENT) {
            addButton_Clicked(event);
            return true;
        }
        if (event.target == cancelButton && event.id ==
            Event.ACTION_EVENT) {
            cancelButton_Clicked(event);
            return true;
        }
        return super.handleEvent(event);
        }

        //{{DECLARE_CONTROLS
        java.awt.Label titleLabel;
        java.awt.TextField titleField;
        java.awt.TextField authorField;
        java.awt.TextField isbnField;
        java.awt.Label label1;
        java.awt.Label label2;
        java.awt.Button addButton;
        java.awt.Button cancelButton;
        java.awt.Label label3;
        java.awt.TextField itemsField;
        java.awt.Checkbox bookButton;
        CheckboxGroup Group1;
        java.awt.Checkbox magazineButton;
        java.awt.Label label4;
        //}}

        //{{DECLARE_MENUS
        //}}
    }
```

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

《分析模式：可重用对象模型》学习笔记之二：责任模式

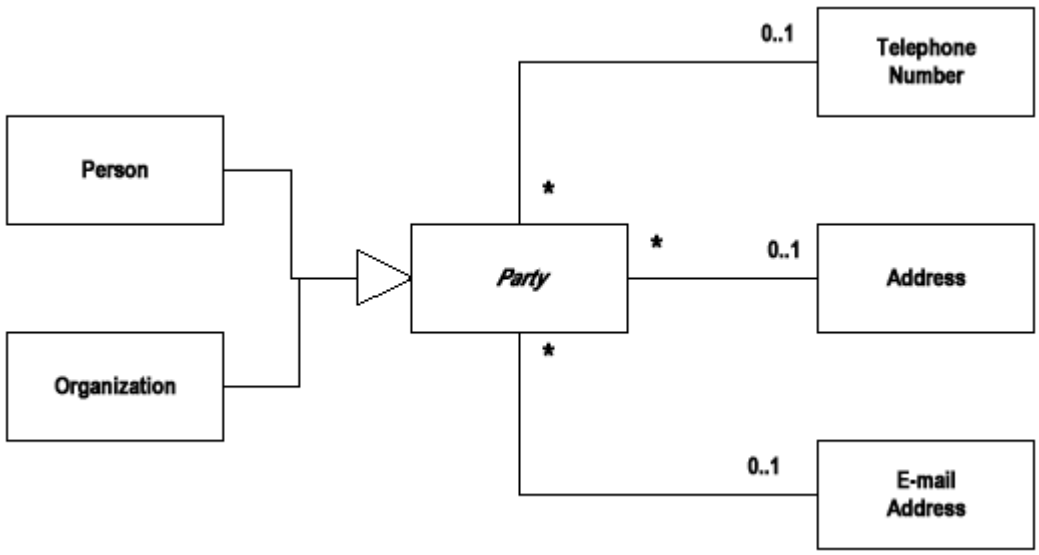
Windy J（windy.j@163.com）

1 责任模式

这一章关注的重点是**关系**，以及怎样为错综复杂的关系建立模型，另外，所有的插图都来自原书（《Analysis Patterns：Reusable Object Models》），并遵循 UML 标准。

1.1 Party 模式

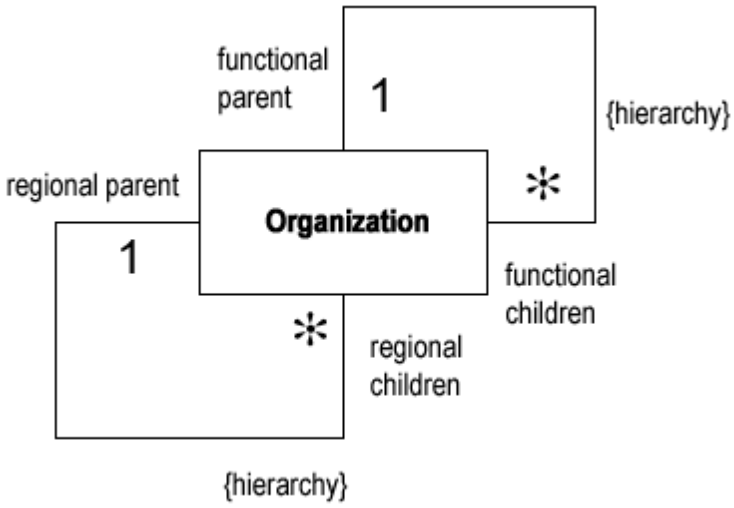
在这一章中，首先我们接触到是是 **Party** 模式，在进行系统分析和概念模型设计的时候，经常发现***人和各种各样的组织有着同样的行为***，例如，固定电话的计费可能是针对个人，也可能是一个单位；需要各种服务的时候，你可能求助于一个服务公司，或者服务公司一个特定的业务员。总之，因为人（**Person**）和组织（**Organization**）表现上的一致性，如下图所见，我们从中抽象出 **Party**，作为 **Person** 和 **Organization** 的抽象父类。



1.2 组织（Organization）的内部结构

第二步，如果我们把注意力转移到组织（**Organization**）的内部结构，就会发现一些有趣的问题，通常最常见的一种结构是金字塔结构，因此建模时可能按照这样的结构建立线性的模型，例如：

这样的模型并没有错误，但是有缺陷，首先不能满足比较复杂的组织关系，更严重的是，一旦***需要更多的层次关系***，例如存在部门直接上下级关系以及区域附属管理方式，必将引起整个模型的更改，对系统的影响可想而知，在这种情况下，最通常的改进措施是引入层次关系，如下图所示：

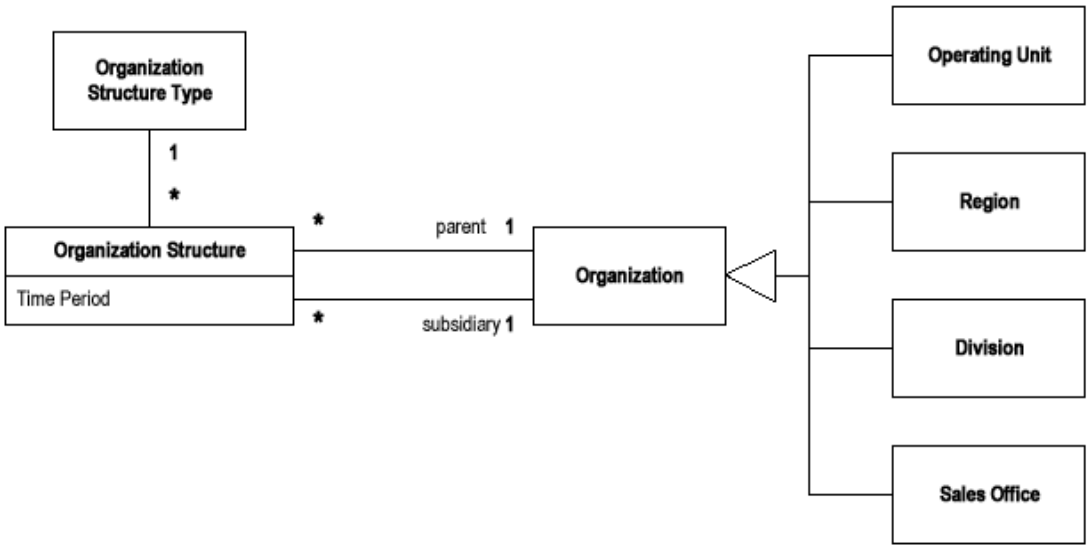


通过增加新的关联关系，可以灵活实现组织（**Organization**）之间的各种关系以及可能的变化。在上图中，**{hierarchy}**是一个约束（**constraint**）来限定关系。

1.3 组织关系抽象

第三步，在一般的情况下，以上的模式已经足够解决问题，但当这样的层次和组织关系很多而且复杂时（超过两种），例如现在流行的矩阵管理，就可以将关系本身抽取出来独立处理，如下图所示，作者此时考虑到组织结构的有效时段，所以加入了一个时间段属性来记录组织结构的存在时间。

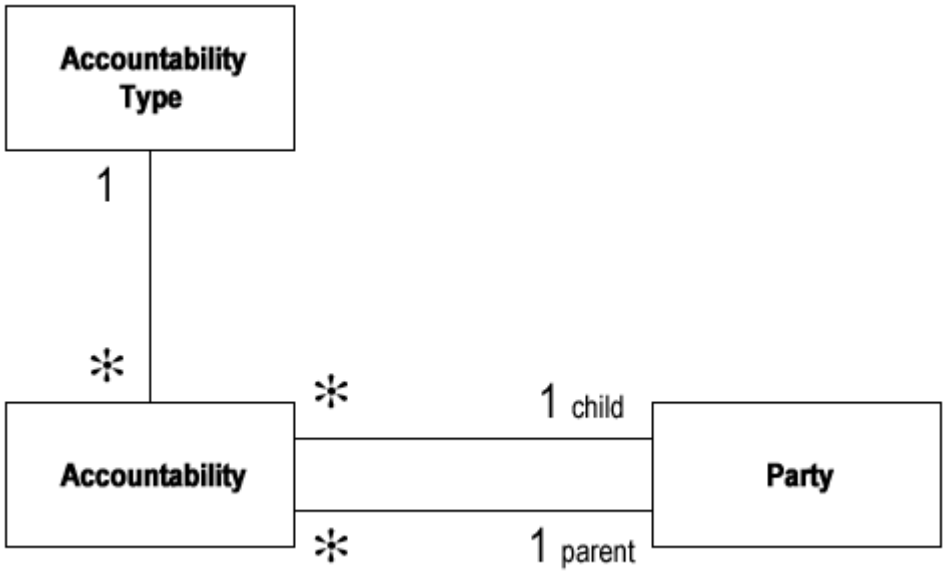
Figure 2.6



请注意，在这个模式中，**Organization Structure** 才是模式的核心，在系统中，由两个 **Organization** 的实例（分别充当 parent 和 subsidiary），以及一个 **Type** 实例来说明该结构的类型。在这样的结构中，可能存在许多的规则（**Rule**），这些规则可以根据情况分别处理：如果 **Type** 很多，而且规则主要跟 **Type** 有关，就分配给与 **Type** 相关联；如果 **Type** 并不多，但主要根据 **Organization** 的子类型变化，就可以分布到 **Organization** 的子类型中。

1.4 责任（Accountability）模式

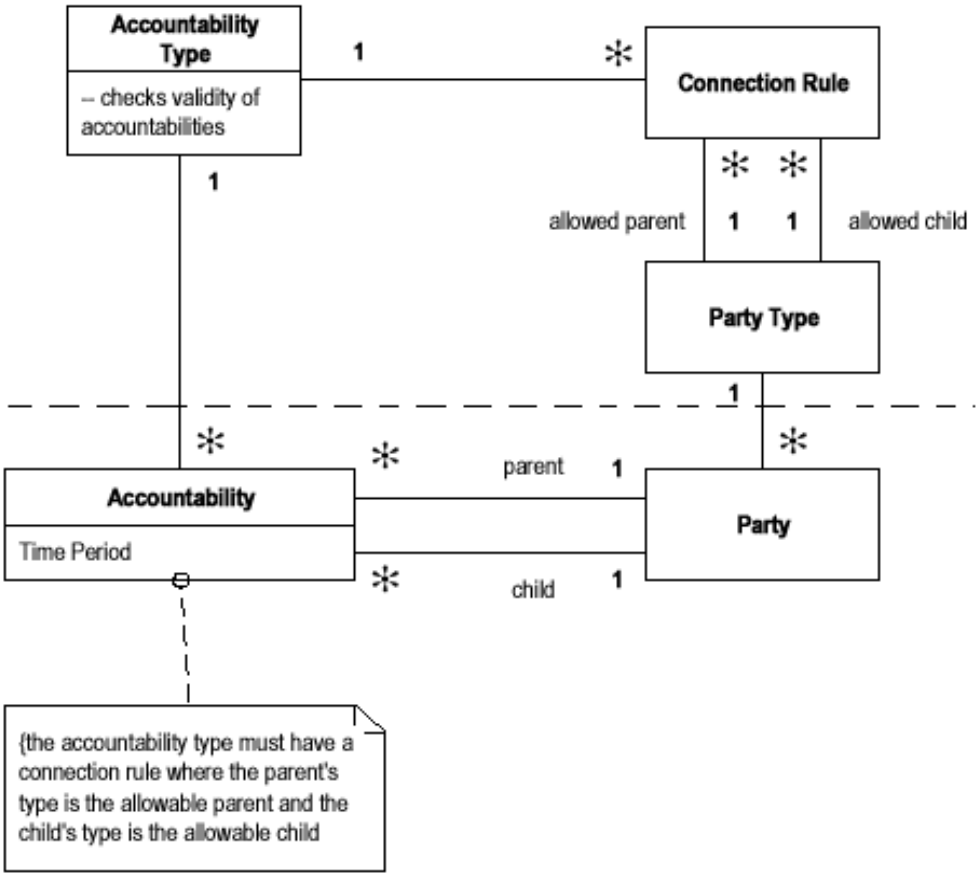
第四步，从第一步看到，**Party** 是 **Person** 和 **Organization** 的抽象父类，因此把 Party 代入上面的模式（有点象我们小时侯代数里常用的代入），正式形成责任（**Accountability**）模式。



1.5 知识层（Knowledge level）和操作层（Operational level）分离

出现这样一种想法是考虑到以下情况：当 **Accountablity Type** 的数量比 **Accountability** 的数量多很多的时候，处理 **Accountablity Type** 的规则也变得更为复杂，要解决这样的问题，就可以引入知识层和操作层的分离。

由下图可见，用虚线隔离开的，就是知识层（**Knowledge level**）和操作层（**Operational level**），在这个模型中，知识层（**Knowledge level**）由三个类协作完成，它们分别是 **Accountablity Type**、**Connection Rule**、**Party Type**，在 **Connection Rule** 中定义合法的 **Party** 关系规则，并通过 **Accountablity Type** 对 **Accountablity** 进行创建时的合法性检验。它的另一个好处就是，可以将知识层的实例化独立出来，作为操作层（**Operational level**）运行时的配置；换句话说，当知识层的规则改变时，系统的行为将被改变，而不需要任何其他代码的改动，这当然是一种比较理想化的情况。



由此想到，构建专家系统的设计思路也可以从这个模式得到一些启发，这是笔者一时的感触。

在原书中，如何实现这样的模型提得比较模糊，但是笔者认为，可以将它们作为正常的模型来实现，两个层次的区分只是表明它们各自担负的任务和地位不同。知识层倾向于描述系统可能存在的各种形式，并设定判断系统是否有效的各种规则；操作层则描述在这样的配置下系统实际的行为。通过改变内在的配置来改变外在的行为，就是这个模式的目的。

由于这个模式的特点，改变系统行为时不必更改操作层的代码，但是，并不意味着改变系统行为连测试也不必要做。同样，也需要调试、配置管理。

作者也提到，这样的模式用起来并不轻松，甚至在一般的系统中也不必要，但当你发现有必要用它的时候，别犹豫（感觉象用降落伞一样）!

1.6 小结

从简单到复杂，前面分五步介绍了适用于解决 **Party** 及其关系的各种模式，每种推荐的模式都有其表现的机会，希望我这篇文章可以起到一些抛砖引玉的作用，并欢迎大家对其中的错误进行指正，欢迎发表意见，进行交流。

原文参照《Analysis Patterns：Reusable Object Models》，Chapter 2，Martin Fowler

参考文章：Party，by Martin Fowler

作者：Windy J

2001/5/10

《分析模式：可重用对象模型》学习笔记之三：观察和测量

讨论

Windy J（windy.j@163.com）

1 观察和测量（Observarions and Mesurements）

许多计算机系统记录现实世界中各种对象的信息，这些信息通常表现为计算机系统中的记录、属性、对象等其他各种各样的形式。最典型的方式是把某项信息记录成某个对象的一个属性，例如，一个人体重 70 公斤记录成“人（Person）”类的体重（Weight）属性，值为 70。本章将讲述这种方式的不足，并提出一些更合理的解决方法。

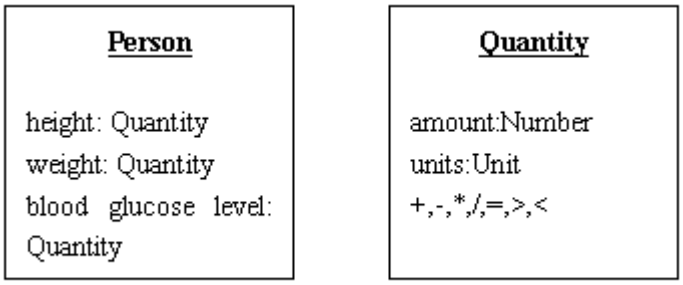
本章的模式来自与医疗领域有关的项目，所以采用了许多这一领域的例子。

本章中的模式图均由笔者以通用的 UML 格式重画。

1.1 数量（Quantity）

当用上面提到的方法记录数据时，最常见的不足之处就是单纯的数字不足以代表它的意义，是 70 公斤，70 磅，或者别的什么？我们需要确切的单位。可不可以创建属性和个单位之间的关联？可以，但那样系统中将会出现错综复杂的关联，从而增加了系统的复杂度。

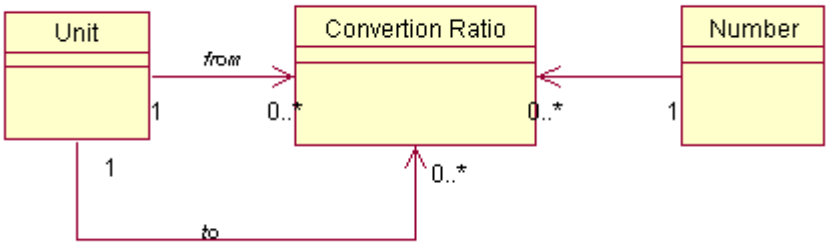
如果改用一个 Quantity 类来表达，这样的意义将会更简单。如下图所示，Quantity 类包括一个 amount 属性，记录数值，一个 units 属性，记录单位，并支持一般的运算操作。



1.2 转换比率（Conversion Ratio）

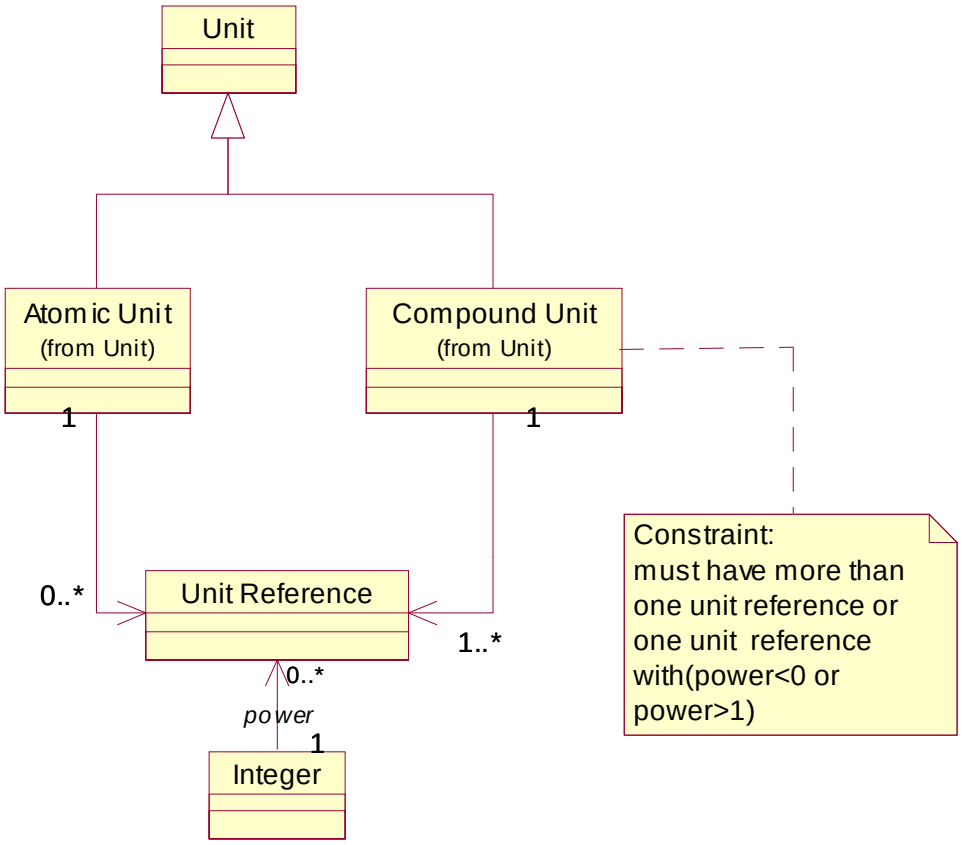
这个模式主要是解决不同的单位间转换的问题，通过需要转换的不同单位，以及它们之间的比率，就可以实现各种固定比率的单位转换，这在系统中也是相当有用的。

但它也有不足的地方，如果转换比率不固定，可能需要额外的计算函数。对于一般情况来说这个模式已经足够了。



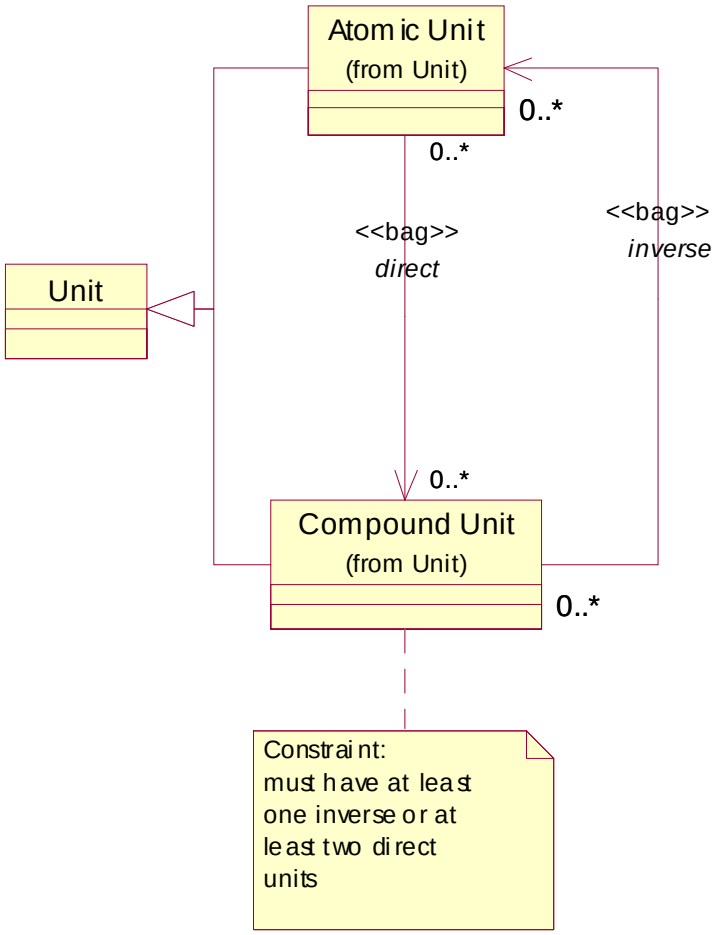
1.3 复合单位（Compound Unit）

单位可以是复合的也可以是不可再分解的（基本单位），如何来表达复合单位？请看下面两个模型：



在这个模型中，复合单位（Compound Unit）通过单位引用（Unit Reference）来记录基本单位（Atomic Unit）和它们的幂（power）。这是一个较直接的模型。

下面这种模型由于使用了 bags，从而显得更紧凑：

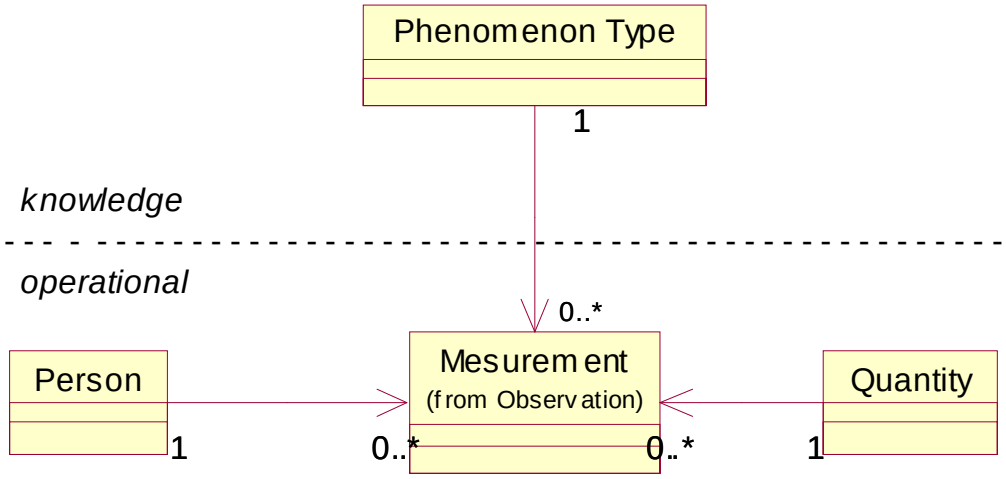


这两种模型差别不大，只是一个采用了 bag，而另一个用了 Unit Reference，至于在什么时候采用这样的模型，要不要引入复合单位，则完全取决于客户和应用的需要。

1.4 测量模式（Mesurement）

当一个复杂的系统中包括成千上万的测量活动，需要记录那些测量数据时，光靠数量模型是不够的。如果还是把测量数据当作属性来处理，系统中可能充斥着杂乱无章的属性，从而导致一些非常复杂的界面（Interface）。这里的解决方案是把各个测量项目（例如医院需要的身高、体重、血压、血糖浓度等）当作对象，并把对象的类型引申成“现象类型（Phenomenon Type）”，这个时候，问题的复杂度就转移到各种各样的 Phenomenon Type，以及各个测量（Mesurement）实例。

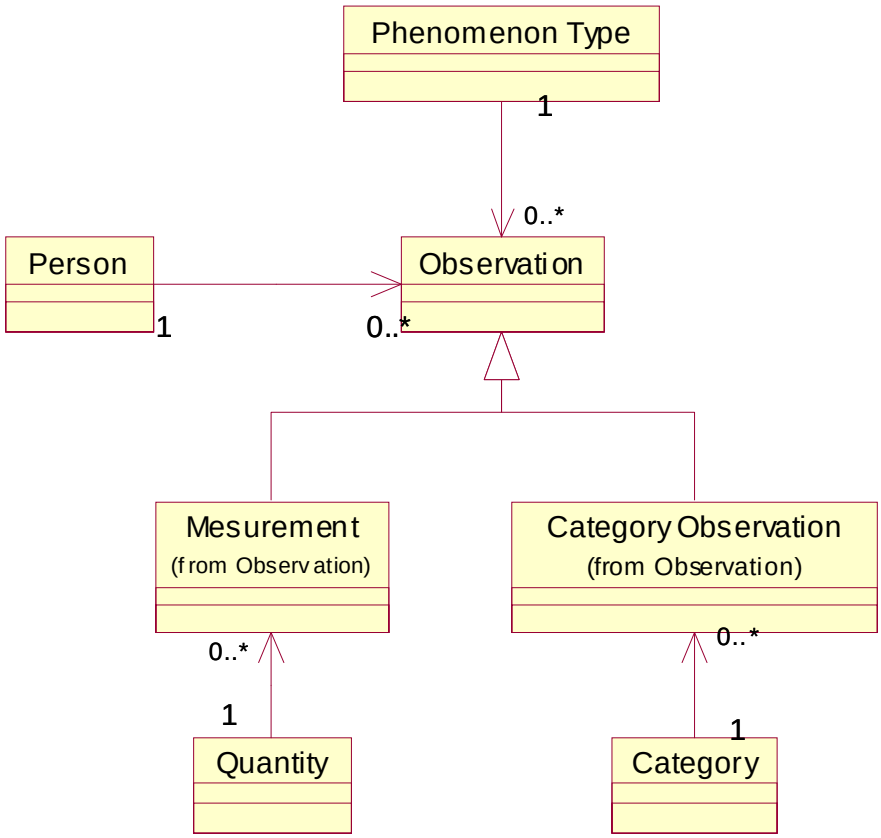
如下图模型所示，测量（Mesurement）实例已经包括了类型（Phenomenon Type），测量对象（Person），结果（Quantity）。依照我们上一章提到的知识层和操作层分离的思想，可以把现象类型（Phenomenon Type）划分到知识（knowledge）层，因为这些对象表示了千千万万要进行的测量项目，这部分信息是相对不变的。



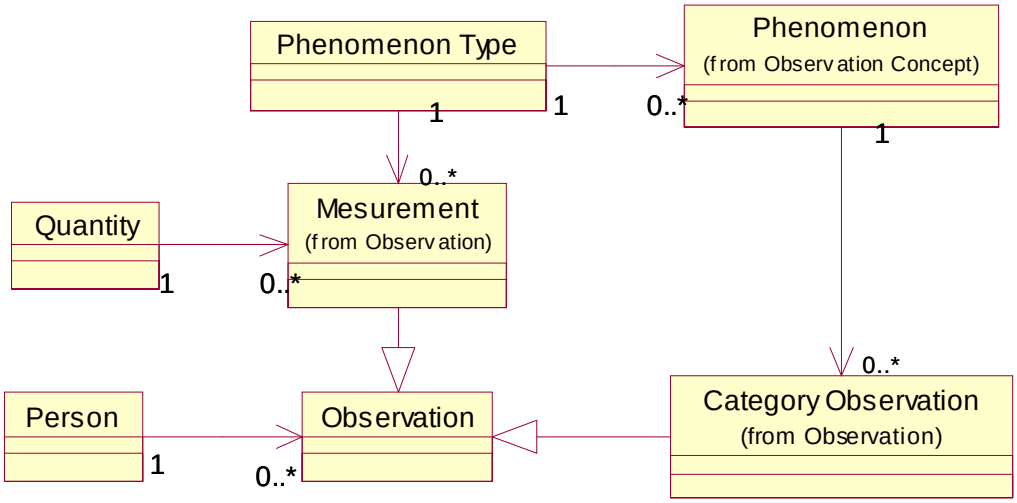
也可以将单位信息跟类型（Phenomenon Type）关联，而测量实例中只保存单纯的数值，但是为了对类型（Phenomenon Type）进行多单位的支持，我们还是倾向于采用上图的模型。

1.5 观察模式（Observation）

由于进行测量时除了要记录许多测量项目具体的数值，例如身高，体重，血糖浓度等之外，还有一种测量项目只需记录条目类别，例如血型，常见的有 A，B，O，AB 四种，又例如体形又可以记录为肥胖，正常，偏瘦等类别，所以我们有必要在上述的模型补充种类（Category），从而正式形成观察（Observation）模式：

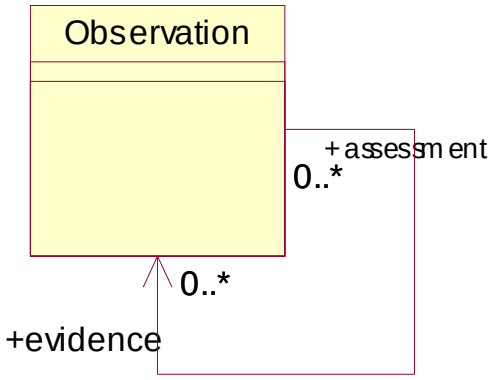


在这里，种类（Category）最好和现象类型保持较为简单的关系，可以减少歧义，降低复杂性。在下面的图中，Category 被改名为 Phenomenon，并被移动到知识层，由 Phenomenon 来定义某些 Phenomenon Type 的一组可能的取值。



对于观察（Observation），还要提到的是，在实际的诊断过程中，医生需要根据某些现象来推断某些测

量活动，而这些测量活动又为其他的测量记录提供证据，所以可能还存在以下的关系：

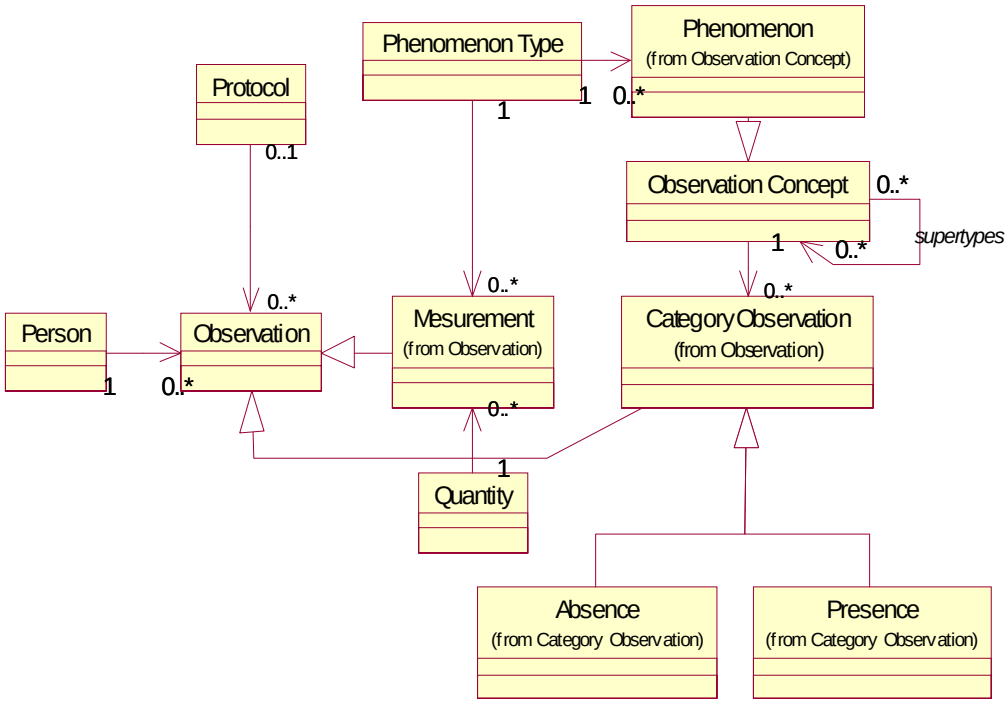


1.6 对 Observation 的完善

在以上的图中，还遗漏了一种可能的情况，那就是对于有的记录项目，结果可能就是简单的“有/存在（Presence）”或“没有/不存在（Absense）”，例如针对各种疾病而言，某个人或许有糖尿病，或许没有。这就是 Category Observation 出现两个子类 Absence 和 Presence 的原因。

在这里，为 Phenomenon 增加了一个父类 Observation Concept，这是为了在处理例如疾病的时候它们可以不必跟 Phenomenon Type 相关联。

还有，可以看到，在 Observation Concept 上加入了一个名为 SuperType 的自关联，是因为各类疾病可能互相存在关系，其中的一种关系是上下级的关系，例如糖尿病和 A 类糖尿病，B 类糖尿病；如果存在 A 类糖尿病，那么一定存在它的上级；如果 A 不存在，并不能表示上级存在与否。



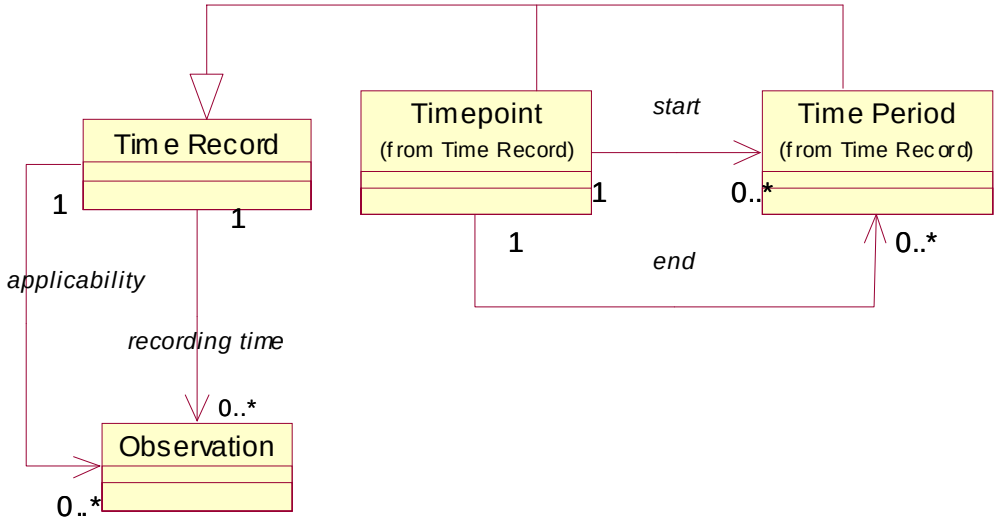
1.7 协议（Protocol）

在上图中，Protocol 是一个重要的知识层概念，表示进行观察时采用的方式。例如我们量体温的时候体温计可以放在腋下或含在口中，某些时候，很有必要记录这些不同的方式。而且，可以根据不同的观察方法判断结果的精确度和灵敏性。

把这部分信息单独放在 Protocol 中，简明而易于处理。

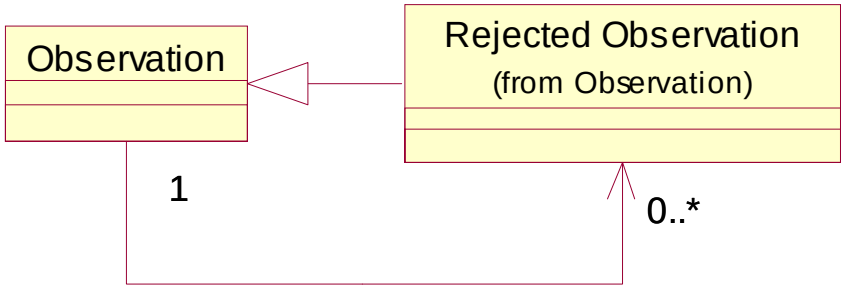
1.8 双时间记录（Dual Time Record）

Observation 经常有一个有限的有效时间，在下面的图中，Observation 跟两个时间（TimeRecord）有关，一个表示记录时间，另一个是发生时间，Time Record 可以同时表示时间点和时间段，这可以由它的子类体现。



1.9 被拒绝的观察

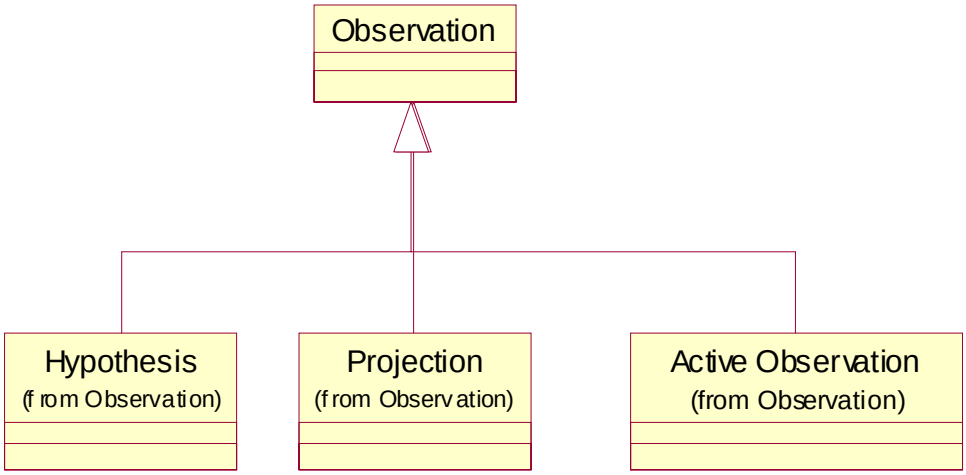
在进行观察时，总有一些对象是被拒绝的，可能是错误的，不合适的，过期的，那么，这部分的 Observation 成为 Rejected Observation，并作为 Observation 的子类。每一个 Rejected Observation 都与一个 Observation 有关，而这个 Observation 正是确定原来的 Observation 成为一个 Rejected Observation。



1.10 主动的观察、假设（Hypothesis）和 Projection（估计）

在医疗系统中，一个医生经常需要根据已有的观察来判断，做些推测和估计，这些也许并不是实际的测量，因此在这里为 Observation 引入可能性，由它产生主动的观察、假设（Hypothesis）和 Projection（估计）三个子类。Hypothesis 需要进行更多的观察来确定，如果 Projection 是真的，则需要更多的主动观察来支持它。

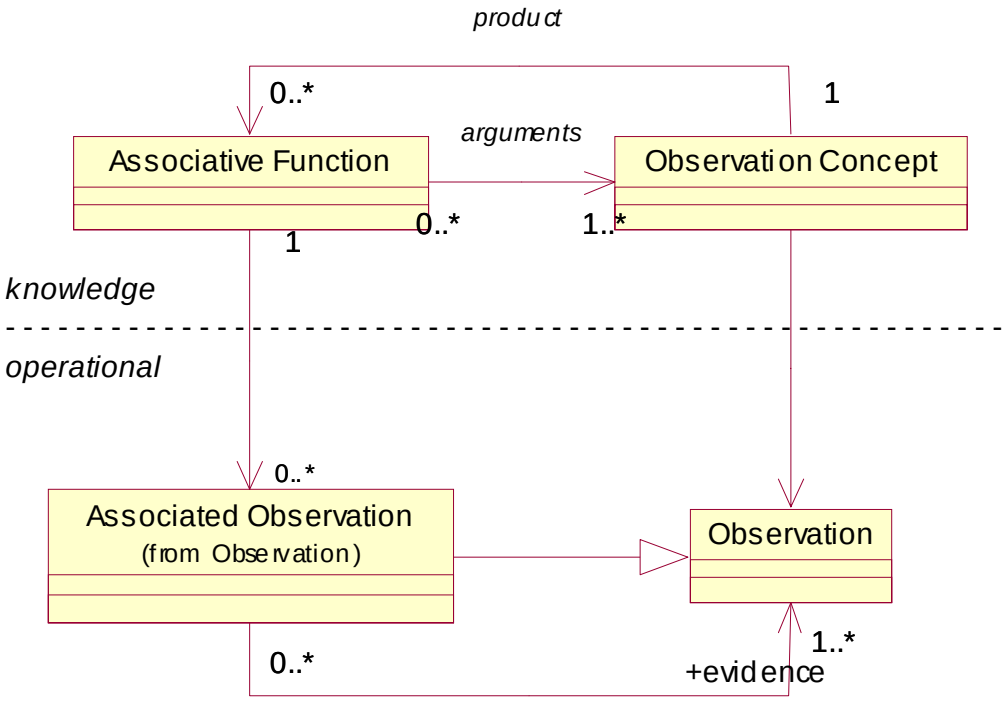
观察结果的确定程度会引起大家的注意，但是这个方面可能由医生来决定比较好。



1.11 观察关联

在一个诊断里往往有一连串相关的观察，在以下的模型里，观察（Observation）可以与观察（Observation）相关联，在知识层 Observation Concept 也要与 Observation Concept 相关联，这样的关联通过在知识层定义 Associative Function，通过 Observation Concept 作为参数和关联的结果，制订观察相关联的规则，从而对给定观察，可以找到与其有关的关联。

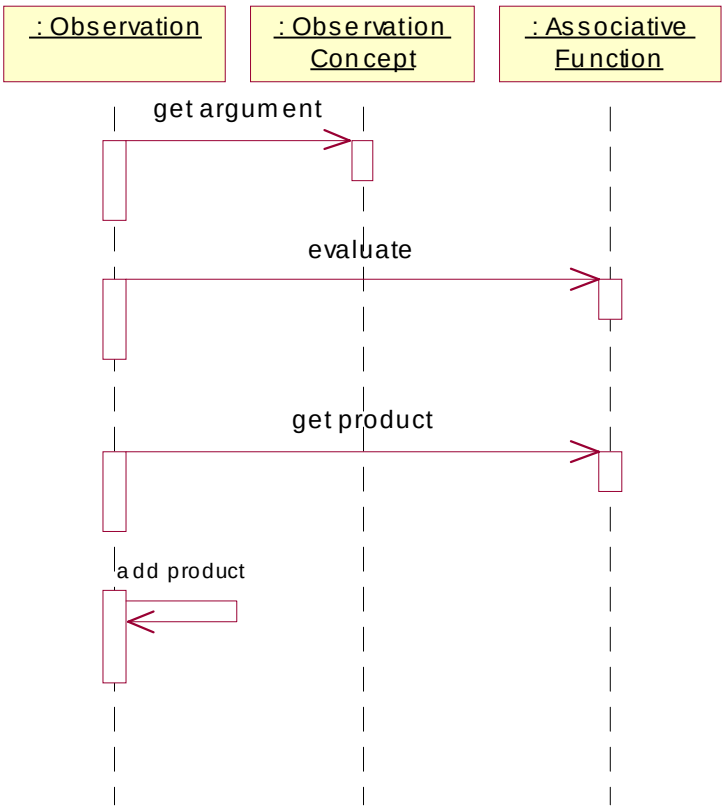
知识层与操作层并非完全形成镜向映射，可以看到，Associative Function 并不是 Observation Concept 的子类。一个 Observation Concept 可能有几个 Associative Function 来得到它作为结果，但一个特定的 Observation 只有一个 Observation 集合作为证据。



1.12 观察过程

有必要再来看看进行观察的过程，一个简要的过程是，从进行观察（可能是某些测量）开始，然后从知识层以及上一节的模型得到与之相关的观察，并把这些观察列入建议在下一步进行的观察中，这是一个循环的过程。

更复杂一点的规则和描述是，从一个观察开始，找到 Observation Concept，查找该 Observation Concept 作为参数的 Associative Function，对于每个 Associative Function 进行验证，得到相应的 Observation Concept 作为 product，将 product 作为答案进行下续的观察，也是可以循环的。可以观看序列图如下：



一个更加详细的观察过程还包括把观察区分为主动的观察、假设（Hypothesis）和 Projection（估计），并对其中的 Projection（估计）和一部分活动的观察进行分析，得到后续的观察，对其中一部分活动的观察根据互相矛盾的观察结果来判定它们是否可被拒绝（Rejected Observations）。

1.13 小结

从常见的数量（Quantity）模型开始，到最后整个观察过程的介绍，这一章的内容包括医疗系统中可以适用的通用模型，其实这些模型并非用于特定的某个领域，如果发现它们跟你的项目情况有相似之处，就可以考虑使用或做一些改动；而下一章，则会介绍观察模式在企业分析中也同样适用。

原文参照《Analysis Patterns：Reusable Object Models》，Chapter 3，Martin Fowler

作者：Windy J

2001/6/8

参考文章：
From Analysis to Design of the Observation Pattern
Observation Model
An Extension and Implementation of Fowler's Observation Analysis Pattern
The Validated Observation Pattern

用户需要什么-软件的工程可用性

讨论

（第一部分）

Larry L. Constantine 著，Huang Yin(yin.huang@ssmc.siemens.com.cn) 译

“究竟用户的需要是什么？”如果 Fred 作为一个程序员而不是一个心理学家时他可能会提出这样一个问题。用户们通常需要更多，而开发人员似乎看上去并不能很好的领会并更好的满足他们。对于我们而言，写出可靠的代码以及为系统加载一些功能看上去似乎是不够的。那么，他们需要什么呢？

用户真正需要的是一个好的工具。所有的软件系统，从操作系统和语言到数据输入和决策支持应用程序，都是正确的工具。终端用户希望能从我们为其设计的工具当中获取得更多，这与我们希望从我们所使用的工具所获取得更多是一样的。他们希望系统易学易用而且能有助于他们的工作。他们希望软件不要使他们的工作放慢，不能欺骗或是迷惑他们，更不能容易出错或是难以完成任务。

这不是个容易的任务。你知道这是很难的，主要的商务软件包经常由于一些显而易见的可用性上的不足而变得费解。甚至最大的软件开发商看上去也是更多地把精力投放在艰难的实现快速变化的特性而不是交付一份有简明的可用性的产品。

并不是行业中认为可用性不重要。新的用户界面窗口小部件和 GUI 开发工具的广告充斥了商业杂志。关于用户界面设计的书在书店里的书柜中排列成行。所有的软件公司都详细地描述他们在昂贵的可用性试验支持下的用户界面测试程序。我们过多的关注用户界面设计--对于一些项目而言几乎是预算的三分之二--但仍然与项目的目标谬之千

标准操作过程

用户界面标准和可用性方针常由于对最好的用户界面的过高期望而造成极大负担。Microsoft，Sun，IBM 以及 Apple 都已经公布了他们的用户界面标准，而且很多公司也纷纷制定出公司自己的内部使用手册来支持已发布的标准。那么这样起了多大作用呢？ 位于 Massachusetts 的用户界面工程学会的 Jerrold Spool 的发现，现有的标准和方针仅能覆盖现实项目中的 10%的问题，定制的内部手册也仅解决了另外的 10%。尽管如此，在这些标准不能够解答开发者所面临的大多数难题。

有一些标准甚至是与可用性相抵触的。假设获悉男性中有十二分之一是具有不同程度的色盲，那么由此简单的规定要所有的用户界面不能有颜色。这样简单的规定有效地使有操作障碍的人得到平等对待，但也限定了用户界面是难辨认且是灰蒙蒙的。

一些过去的错误中的遗留物永存于标准中，因而促成了一些 构想笨拙的控件，例如滚动栏（见 1994 年 8 月的 *Windows Tech Journal* 中的“图形导航”）。或在 Win3.X 中想出 10 种关闭标准应用程序的方法。和标准一致固然好，但是武断的一致或是错误的一致并不能制造出更有用的软件。

里。肯定在某些地方出了问题。

我们可能已经论证过我们生产肥件(fat ware)的能力，但不是我们生产出更有价值产品的能力。我们在项目重压之下作出的很多好的程序代码也许会因为大多数用户今后仅使用我们费尽心思做出的产品的极小一部份而宣告浪费。奇怪的是很多这样的功能在一堆控件，菜单，对话框之中根本不可能找得到。我们结束对那些将来没有用的特性的咒骂，并且留下有用的部分。我们也由于将简单的任务复杂化而遭到谩骂。

我们的一些为改进这种状况的所作的努力可能仅仅是使事情更糟。我们急切的使用最新的 3D 直观模型，却只得到不领情的用户的埋怨。他们读不懂我们的一个个灰色的对话框。我们用密密麻麻的带图标的工具条来告诉他们这里有很多功能，但是他们觉得这很难理解。可能当我们在第一步做了一个简单的界面之后我们会设计出改进了的界面向导来使复杂的界面易于使用。

一个不漂亮的事实是可用性并不是基于这些吸引人的图形、改进了的可视化控件或是活动的 Agent(译注：可能是指 MS-Office 里的 Agent)之上。你不能从 3D 直观图，彩色图标或是浮动工具条中获取可用性。这不仅仅涉及你使用了什么窗体控件，还包括你如何使用这些控件，以及这些控件如何整合在一起工作。

最终，可用性来自于用户界面的体系结构迎合了用户所希望的实现。这意味着你不得不通过了解用户的工作来使你的软件迎合要求。这也意味着你必须设计出整个用户界面的体系结构。这包括了用于支持这个工作的所有的结构细节以及动态行为。

为了做到这些，为了在软件中设计可用性，你需要工具。你需要工具来弄清用户需要做什么和他们需要什么来支持他们工作的进行，你还需要工具来组织复杂的用户界面的体系结构而不遗漏掉任何细枝末节。或者说，你正准备好去竭尽全力制作出一起被提交的程序，你需要能够集中你的注意力，有效的利用你的时间，因为当用户抱怨时，老板会逼着你在规定期限内交付新程序。所以，你也需要一个简单的灵活的工具来组织你的工作。

这篇文章介绍了一些用于制作更好的（用户用的）工具的工具——简单的用于制作出能很好的满足用户需要的小而简单的系统的方法。这当然不可能是一个完整的故事，也不能使你变成一个可用性的专家。但是这可以给你一些比较前卫的想法而使你的下一个项目有真正的改观，帮助你一开始就能制定出可用性而不是中途应要求突然改变或是最后的时候再将它加进去。

围绕可用性的设计

用户和用户界面的问题并不总是发生在现在。在一开始，并不存在用户，只有操作员以及只是疯狂操作电脑的程序员，他们反复操作开关并观察着操作台上的灯，并不存在给用户的真实界面。有打孔卡片，打孔带，还有打印机。你把卡片或带打上孔并不断送至读卡机，并将打印机里打印的每一页存储起

来，终端用户就获得有一行行记录的报告。其中的一些内容被格式化，还被排列成多少有点可读性的序列。

技术人员更注重的是技术而不是人，因此程序常在发觉用户的存在之前就形成用户界面。这样注意力被集中在技术问题上，如屏幕的绘制和域长，数据确认以及退出键的设计。偶尔也曾意识到用户，就是真正的人，正坐在系统另一端注视着屏幕，而且敲击着功能键。也许轻视和忽视用户的症状由来已久，专业通行了一种有友好的用户界面，却有平淡无味的交互的主流，往往就形成了覆盖在同样陈旧的狭隘和顽固的编程方式上的一层薄纱。

为什么要问“为什么”？

围绕可用性的设计是软件开发的一个相关的新的走向，它始于一个简单的问题：为什么？为什么这是必要的？为什么用户要与这个系统交互？他们想要做到什么？

为何要问“为什么”？因为只有询问了用户为什么要用这个系统之类的问题才能帮助我们关注那些有关系统可用性的基本要点。

为什么有人要用在线的电话目录。因为他们要联系那些他们不知道或是忘了联系方式的人。关键的设计要点是在不完全或是不精确的信息的基础上有效的设定一个系统入口。一个好的用户界面将以此作为焦点，减少操作步骤来找到号码。为什么用户要把银行卡插入自动取款机？是为了鉴别身份。带有磁性条纹码的卡是送到终端的一种方式，另外还有声波调用辨别或是红外线扫描也能达到这个目的。为什么在文档中用户要调用一个对话框用于输入一些特定的字符？因为一些符号或是字符在键盘上不存在。这个对话框会显示一些用户希望得到的字符，而不是在键盘上存在的字符。以可用性需求和被标识出的焦点事件来开始一个项目将可以节省开发时间。用户界面的体系结构将根据这些要点组织起来。开发人员可以避免在调整对可用性影响极小的特性上浪费时间，而关注更为重要的事件。

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

深刻理解 CMM－成功的关键

讨论

蓝尔公司

“SEI的一项调查显示，对 CMM 各级别转变的关键因素的深刻理解，是 CMM 实施成功的关键”。纵观国内外实施 CMM 的企业，成功者有，不成功者也为数不少。ISO9000 在我国实施的情况，从企业到主管质量的上层领导，都深有感触，我国企业目前获得 ISO9000 认证证书总数已超过 25000 张，是亚洲第一(日本及韩国也只有 1 万多家)，然而，残酷的现实表明，我国并不是质量强国，面对着“亚洲第一”这一桂冠，主管质量的上层领导忧大于喜，企业也很迷惑，为什么同是国际管理标准，国外企业采用很成功，而我国企业采用却感觉不到效果－－理解，深刻理解 ISO9000 内涵，是成功的关键，然而，现在能真正深刻理解 ISO9000 标准内涵的审核/咨询人员，实在太少了。我们按三种类型的采用模式对通过认证的企业剖析如下：

采用模式(按 94 版)	企业类型	理解程度	效果/体系成熟度
第一类 ISO9001/2+ISO9004+总部要求(现代管理理念---第四代管理理念及方法)	大多为优秀的跨国公司	总部的要求就是深刻理解的具体要求(实际上此标准就是这些优秀跨国公司成功经验的总结)	每年都有具体的目标/指标，在提高质量的同时，提高工作效率降低成本。 体系成熟度：3 级以上
第二类 ISO9001/2+ISO9004	合资企业及总部力量不强的独资企业或跨国公司，为数不多的国内企业(与咨询老师有关和受顾客的影响) 此类企业同时也采用 QS9000、VDA6.1　6.3	能够一般性理解，94 版的指导性仅仅如此，2000 版溶合了现代管理理念，指导性更强。QS9000、VDA6.1　6.3 对过程改进的指导性较强	不会增加成本，部分企业的工作效率有一些提高。 体系成熟度：2 级至 3 级之间
第三类 ISO9001/2	国内企业(民营、国营)管理基础较好，工作效率高的中小型民营企业及国营企业	基本上是字面上理解，为了满足一些咨询/审核人员要求，又不影响自身的工作效率，有时要用真假两套。	成本增加 无明显帮助 基本上仍是： 高质量----高成本 当成本的承受能力有限时，只好降低质量。 体系成熟度：2 级或不到 2 级

	国内企业 管理基础较差，职责权限不清的企业	字面上理解 能按咨询/审核人员要求做，将职责权限分清	成本略有增加 一开始能感觉到对企业的管理有明显的帮助，但以后便无太多的帮助，停留在原地，成本不会降低，工作效率不会提高。 体系成熟度：2 级或不到 2 级

注：1. 以上三种类型企业的剖析，是我们所接触过企业的整体感觉，并未做具体的数据调研，可能有些判断有误，仅供参考，如有不同的见解愿意与我们交流者可与我们联系，info@laner.com.cn 共同探讨。

2. 体系成熟度是按 ISO9004：2000 版标准中 5 个级别区分 （与 CMM5 个级别相似）

ISO9000 标准实施的经验教训告诉了我们不成功的深层次原因----是字面上理解(知其然，不知其所以然)，还是用现代管理理念深层次地理解其真正的内涵(知其然，要知其所以然)。所有的管理标准都仅是告诉要这样做，但是为什么要这样做及如何做没告诉，而要达到效果性的实施，就必须要弄清楚为什么要这样做。ISO9000 是如此，CMM 也同样如此。

CMM 是一个很好的管理标准，它是西方成功软件企业从不成熟到成熟整个演绎过程(4 个阶段)经验的总结，因而对软件企业过程改进的可指导性极强。和其它管理标准相比，它对组织结构的规定、形象地体现朱兰三部曲的 5 个级别的模型图(软件过程的可视性(内部活动)—成熟度等级所指示的过程能力(活动结果))、从 2 级---5 级各级关键域按输入—输出方式，PDCA 模式的描述，都是其它的管理标准无可比拟的。我国的软件企业确实是非常需要这种管理模式的帮助，但如果仅从字面上理解 CMM，是能做到符合性---CMM 二级，要达到三级以上很难---根本上不去，因为，达到三、四级的符合并不等于有效，无效果的符合等于不符合，这一点与 ISO9000 不同(ISO9000 审核人员大多数是只查符合，不查效果，所以对企业的帮助不大)，而 CMM 推行的目的是使企业在提高软件质量的同时，降低成本，提高工作效率。二级是给企业做规矩，符合就可以，而三级以上才是标志着企业的过程改进达到此效果。因此，CMM 三级以上，才是我们追求的目标。

和其它管理标准一样，CMM 告诉了人们在各级别关键域要做些什么，但是为什么要这样做和如何做没告诉。

为什么要这样做？--------要深刻理解 CMM 就要知道为什么要这样做，“第四代管理”(美国)可帮助我们理解 CMM 的内涵，它用简单的日常生活/工作中案例说明了难以理解的朱兰三步曲的原理以及在我们日常工作中常遇到的难以解决的问题。的本公司的 CMM 培训课程就是用第四代管理作为辅助教材，帮助学员理解 CMM，取得较好的效果，但要让学员在几天的时间里就完全转变观念和思维方式，一下就全盘接受，是不太可能的，它似一杯浓醇的香茶，有一个品味过程，在理解的思路教给了学员后，我们相信在日后的工作实践中他们是能够品出其真正的内涵——CMM 就是第四代管理思想和方法在软件行业的具体体现。

如何做?------印度 Infosys 公司在 CMM 实践中总结出的经验和其作业文档，给了我们最好的帮助，

是本公司培训的重点教材。Infosys 公司是一家非常优秀的企业，成功地从 ISO9001 Tickct 直接过渡到 CMM4 级(目前已达 5 级)，其 ISO9001 体系参照了 ISO9004，很早就培养了很多能深刻理解 CMM 内涵，并灵活利用管理心理学，人体工效学，统计学的内部 CMM 专家，他们针对本企业实际情况，按 CMM 关键域要求进行过程改进，并将这种改进按项目的模式进行(Infosys 的项目管理基础很好)，当该公司有了商业方面和其它方面要求时，才全盘采用 CMM 模式并成功转到 CMM4 级，我们认为，这是采用 CMM 最聪明的途径-----以较小的费用换取最大利润。CMM 现在我国的软件业很热，可我国还是发展中国家，国家不富裕，企业也不富有，该花的钱是应该花，但要花在刀刃，要看价值所在，要看投入产出比。当看到国内众多软件企业正轰轰烈烈进行 CMM 二级评估(评估的费用昂贵)，我们想，大概美国人在笑。

以人为本，预防为主，是众多管理标准所遵循的管理思想，CMM 也同样体现了这一第四代管理的主导思想。由于软件企业的特殊性:1>软件的复杂性和各模块间的相互依赖性是其它的产品所无法比的，它不遵循著明的 8020 规则，是 100 的符合。2>软件作为产品，是用人的大脑生产出来，不是机器制造出来，因而，软件过程对人的依赖性极大。制造业按人、机、料、 法、环 5 个影响质量的因素分析，人的因素占了 20%(戴明博士认为属于个人责任仅占 4%)，而软件业，如果不采用工具和一些好的方法，基本上完全取决于人的因素，而人的承受能力是有限的，方法和工具的采用，可以减轻一些对人的压力，尽管 CMM 只标识出有效软件过程的特征，而没有谈及对于成功项目来说是最重要的两个因素：人和技术，但要成功实施 CMM，就必须要考虑人与技术，CMM 在不同的级别的区域中给出了不同层次的要求，当过程效率的提高已到人的极限时，则只能用工具及一些有效的方法来尽可能减少“低级”错误的发生，减轻过程对人的依赖程度。管理就管是人，过程思维的方式是一种观念的转变，要给人以宽松的环境，当出现问题时，是人的因素影响，但并不全是个人责任的思维方式---出现问题不是追查个人的责任，而是检查过程，查深层次原因---问题的根源，降低成本，首先要找到成本产生的根源。要做到 CMM 三级以上，除了一些工具、方法的适当采用之外，要用管理心理学、人体工效学、统计学等多方面的知识，并遵循过程思维方式，由于软件企业都是高素质人才，有时可能就是在打心理战。

蓝尔公司为了帮助软件企业深刻理解 CMM 达到三级以上，将分阶段举办 CMM 培训班，由浅入深，近期举办的培训，均为第一阶段的入门培训，帮学员深刻理解 CMM 内涵，了解第四代管理方法(还要在实践中慢慢品味)，及 CMM2 级关键域的具体要求及软件工程方面的知识，学会推行 CMM 第一步要做的事(过程定义、优先改进领域的确定及改进)。以后将举办较为深层次的 CMM 培训。各阶段的课程内容我们会逐步登在www.laner.com.cn 。

CMM 是企业内部管理的有效工具，帮助企业达到以最少的投入获得最佳效果的目的，但由于不涉及企业的经营战略和产品的定位等，即使企业内部的运作达到 4--5 级，软件项目能成功，但并不能保证企业一定成功，只有当内部运作与经营战略均为最佳，企业才能成功，若要持续地保持，则可用均衡计分卡来进行企业的全面自我评价，缺陷之处高度引起注意，查深层次原因并尽快改进，以保证企业有持续的竞争能力。

国力的竞争就是企业的竞争，祖国的强盛期盼着一批优秀的民族工业，由衷的希望 CMM 能帮助我国的软件企业走进优秀之列。

深刻理解+实实在在去做=成功

上海蓝尔科技有限公司 CMM 培训计划(6 月－8 月)

CMM 是一个能指导软件企业高效而有效工作的管理模型，本公司举办的 CMM 培训班，追求培训效果，采用互动式的培训方式，教会学员如何做。已办的两期的培训班，均以优质的服务，赢得学员们的好评。然而，我们还在不断地总结经验，设计更合理的培训课程、案例、及更深层次的情景练习,以达到更好的培训效果。

针对目前软件企业的现状和学员们的要求，我们制定了一个较为完整的培训计划，以满足不同层次培训的要求。

1)帮助全面实施 CMM 的企业达到 3、4 级。

全部的课程内容都是必须的。

2)用 CMM 中的一些方法帮助软件企业提高工作效率，降低成本。

如培训计划中：

序号 3 过程定义培训-----对于一些管理基础较好的企业，本课程可教会其识别关键控制点，将现有流程中不合理之处进行改进。

序号 4 需求分析培训-----需求阶段的改进可提高工作效率 100 倍以上。本课程培训教师实际操作经验丰富，用案例分析帮学员掌握需求收集、分析的技巧和描述的方法，以及需求跟踪和相对应的测试计划的制定等。

序号 5 同行评审培训-----同行评审是一个能高效而有效识别缺陷的方法，”能导致较高的生产率和高质量的软件产品，是作业过程中不可缺少的一部分”。本课程采用团队教师按 Infosys 公司的方法从心理上训练学员，使其掌握有效的同行评审的方法。

序号	课　程	课程内容	参加人员	师资背景及形式	时间/费用
1	CMM 宣贯 一天	1. CMM 产生背景，发展趋势 2. CMM 标准概述 3. 如何用第四代管理方法理解 CMM 内涵 4. 如何用 CMM 方法提高工作效率，降低成本 5 实施步骤	软件企业最高管理层技术总监	管理咨询专家、讲座、讨论	6 月 23 日 7 月 28 日 8 月 31 日 费用：400 元/人
2	CMM 实施(Infosys 公司 ISO9001 转四级经验及案例) 三天		管理层、项目经理、技术骨干、QA 主管	管理咨询专家、讲座、讨论、模拟练习	7 月 19 日、20 日 21 日 8 月 23 日、24 日 25 日 费用：1600 元/人
3	CMM 专题培训 (一)过程定义 二天	1. 过程概念 2. 过程定义模拟训练 (CMM 2 级，3 级) 3. 按生命周期定义过程模拟训练 4. 如何确定最佳过程模拟训练	管理层、QA 主管 项目经理、技术骨干等 (已经过 CMM 二级培训) 以企业为单位，上门培训效果更好	管理咨询专家以模拟练习 (Infosys 公司案例分析)	6 月 29 日、30 日 8 月 27 日、28 日 费用：1200 元/人
4	CMM 专题培训 (二)同行评审 二天	1. 同行评审方法及意义介绍 (Infosys 公司方法) 2. 同行评审三种角色心理分析 (如何消除潜意识因素)	项目经理、软件开发人员、程序员	计算机技术专家、管理咨询专家、管理心理学专家、 模拟练习为主	8 月 20、21 日 费用：1500 元/人

		1. 同行评审模拟练习(读者，作者，协调员) 2. 代码评审模拟训练			
5	CMM 专题培训(三) 软件需求分析 三天	1. 软件需求分析原理(自行开发的和由客户委托开发的)及 CMM 三级要求 2. 如何进行有效的需求分析及准确的规格说明书描述(案例分析) 3. 需求修改的管理(简略) 4. 需求跟踪(案例分析) 5. 测度计划的编制(案例分析) 6. 有关工具的介绍	系统分析员、项目经理、软件开发人员	CMM5 级企业内部培训教师(摩托罗拉公司) 讲座、案例分析	6 月 28 日、29 日 30 日 费用：2000 元/人
6	CMM 专题培训(四) 软件工程 三天	一. 需求管理 1. 需求变更 二. 项目管理 1. 项目策划(WBS 方法，PERT 等工具，资源成本，计划样本) 2. 项目估计(功能点估计 COCOMO 模型，Delphi 方法) 3. 风险管理(风险识别，估计，策划，解决) 4. 项目控制(数据采集样本) 三. 软件质量保证 1. 质量概念和模型介绍 2. SQA 策划(计划样本) 四. 软件配置管理 1. 配置标识 2. 变更管理 3. 版本控制 4. 配置审核 5. 配置状态报	软件技术人员、项目经理、SQA	清华大学郑人杰教授 讲座、讨论	报名预约 日期待定 费用：1600 元/人
7	统计技术 二天	1. 统计技术原理 2. 统计技术方法讲解 3. 共同原因与特殊原因的识别及原因分析 5. 案例分析	SEPG 组成员、SQA 成员	管理统计学专家、管理咨询专家、讲座、案例分析、练习	报名预约 日期待定 费用：1000 元/人
8	质量成本 一天	1. 质量成本控制理论 2. 预防成本与失败成本平衡点选择 3. 案例分析	SEPG、SQA 成员、财务主管	质量管理专家 讲座 案例分析	7 月 20 日 费用：600 元/人
9	团队精神培训 (组间协调同行评审) 二—三天	1. 以游戏的方式使学员感悟到团队的凝聚力和活力带来的效果，并识别出自己的缺陷 2. 以企业实际情况为案例进一步训练学员的团队精神	以企业为单位，上门培训	教育心理学专家、管理咨询专家、游戏、模拟练习	报名预约 日期待定 费用面议

		及协作精神 3. 同行评审的训练			
10	CMM 高级别实施答疑 二天	由 Infosys 公司质量副总裁主讲	管理层人员、SPEG SQA 成员	Infosys 公司质量副总裁 讲座、讨论	报名预约 时间待定 费用待定
11	软件企业管理者 领导艺术 (MBA 课程) 三天	1. 管理技巧 2. 员工满意度、心理分析 3. 影响员工忠诚度因素分析 4. 如何有效地激励人	正、副总经理、项目经理、技术总监	中欧管理学院 MBA 老师 (摩托罗拉公司与 Infosys 公司协助) 讲座、讨论	报名预约 日期待定 费用：1600 元/人

备注：以上均为上海班。

北京班与深圳班可预先报名，日期待定。

培训有关的具体事项在培训日期前一周内发正式培训通知。

以上培训科目均颁发培训经历证明；

外地学员可介绍住宿和购回程票，请事先联系；

为了保证培训效果，学员人数将控制在 30 人以内；

报名地点：上海市虹桥路 333 号(上海慧谷高科技创业中心) 314 室

电话：021-64874271. 021-54631017 传真：021-64874271. 021-54631017

网址：laner.com.cn Email:info@laner.com.cn

联系人：赵雅萍

开户银行：招商银行上海分行徐家汇支行

账号：096929-28819317001

上海蓝尔科技有限公司

2001 年 5 月

回执

单位：

姓 名	职务/职称	参加培训课程名称	联系电话	传 真

听课企业情况调查表

1) 企业规模：	□ 20 以下	□ 50 人左右	□ 100 人以上
2) 管理现状：	□ 未建立体系	□ 已通过 ISO9001 认证 (未参照 ISO9000: 3)	□ 已通过 ISO9001 认证 (参照了 ISO9000: 3)
3) 采的工具及方法:	□ 建模工具	□ 配置管理工具	□ 构件
4) 本企业涉及领域：	□ 人众市场产品软件	□ 企业解决方案	□ 专业软件服务
5) 已接受过的培训	□ CMM 宣贯	□ CMM2 级，3 级标准	□ 软件工程

注： 1) 可选择培训课程

2) 为了使课程内容尽可能符合听课企业的实际情况，请在调查表□ " 上打钩

3) 已参加过本公司培训的企业可不用填写调查表

项目管理规范-RUP 管理实施

（第一部分）

讨论

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

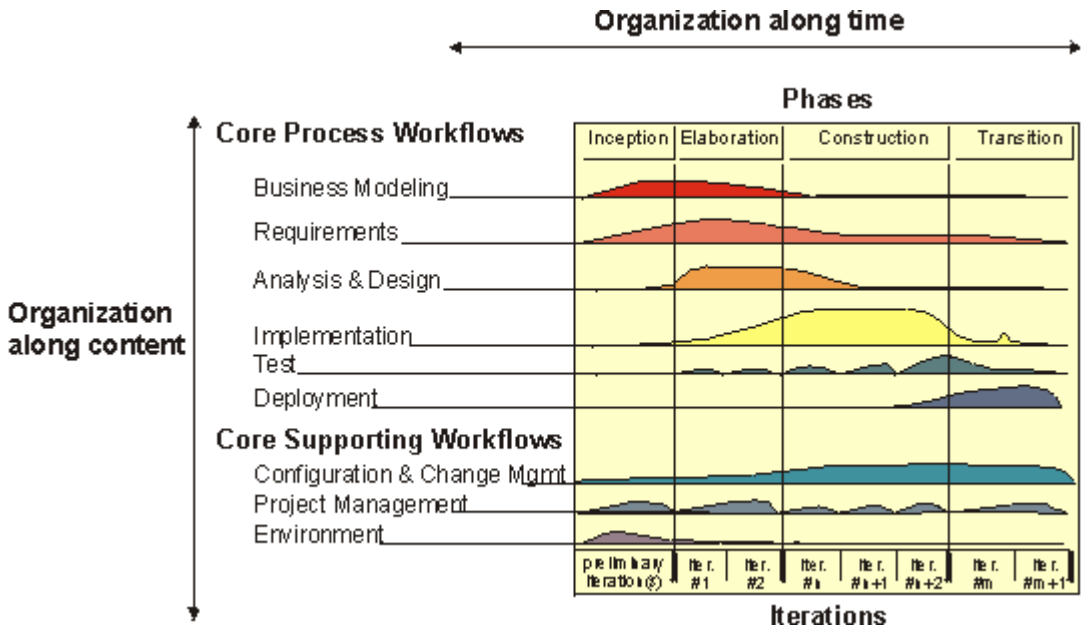
2001 年 3 月 14 日

RUP 简介

Rational Unified Process（简称 RUP）是一套软件工程过程，主要由 Ivar Jacobson 的 The Objectory Approch 和 The Rational Approch 发展而来。同时，它又是文档化的软件工程产品，所有 RUP 的实施细节及方法导引均以 Web 文档的方式集成在一张光盘上，由 Rational 公司开发、维护并销售，当前版本是 RUP2000。RUP 又是一套软件工程方法的框架，各个组织可根据自身的实际情况，以及项目规模对 RUP 进行裁剪和修改，以制定出合乎需要的软件工程过程。

RUP 吸收了多种开发模型的优点，具有很好的可操作性和实用性、从它一推出市场，凭借 Booch、Ivar Jacobson、以及 Rumbaugh 在业界的领导地位、以及与统一建模语言（Unified Model Language，以下简称 UML）的良好集成、多种 CASE 工具的支持、不断的升级与维护，迅速得到业界广泛的认同，越来越多的组织以它作为软件开发模型框架。

在 RUP 中，软件开发生命周期根据时间和 RUP 的核心工作流划分为二维空间。



如上图所示，时间维从组织管理的角度描述整个软件开发生命周期，是 RUP 的动态组成部分。它可进一步描述为周期（Cycle）、阶段（phase）、迭代(Iteration)。

核心工作流从技术角度描述 RUP 的静态组成部分，它可进一步描述为行为（activities）、工作流（workflow）、产品（artifact）、工人（worker）。

图中的阴影部分描述了不同的工作流，在不同的时间段内工作量的不同。值得注意的是，几乎所有的工作流，在所有的时间段内均有工作量，只是大小不同而已。这与 Waterfall process 有明显的不同。

RUP 采用 Use Case 的概念，把要开发的系统根据各功能使用的情况划分多个 Use Case，并采用迭代的思

想把系统的风险分布在四个阶段，风险越大的迭代越要放在靠前的阶段做，使软件产品的风险不断降低；而不是像传统软件工程那样越往开发的后期问题越多。所以 RUP 的思想一推出就受到软件企业的欢迎。按照 RUP 的开发模式一般可以达到 CMM2、3 级的水平。当然，理解和掌握 RUP 需要一个相对较长的过程。

※※

1. 项目阶段

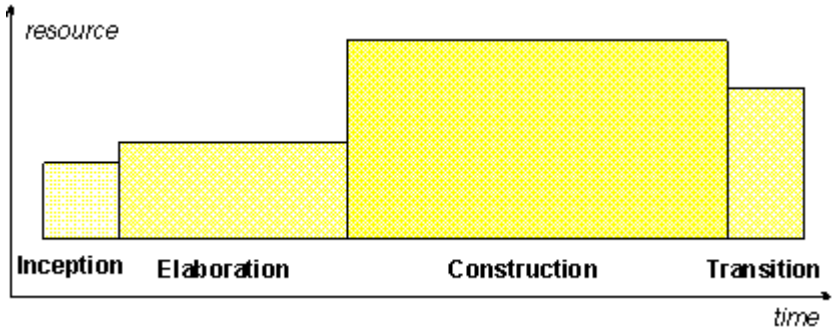
从管理的观点来说，软件生命周期随着时间分为四个依次进行的阶段，每个阶段的结束都有一个主要里程碑；实质上，每个阶段就是两个主要里程碑之间的时间跨度。在每个阶段结束时进行评估，以确定是否实现了此阶段的目标。良好的评估可使项目顺利进入下一阶段。

1.1. 计划阶段

在进度和工作量方面，所有阶段都各不相同。尽管不同的项目有很大的不同，但一个中等规模项目的典型初始开发周期应该预先考虑到工作量和进度间的分配：

	先启	精化	构建	产品化
工作量	~5 %	20 %	65 %	10%
进度	10 %	30 %	50 %	10%

可表示为下图



对于演进周期，先启和精化阶段就小得多了。能够自动完成某些构建工作的工具将会缓解此现象，并使得构建阶段比先启阶段和精化阶段的总和还要小很多。

通过这四个阶段就是一个**开发周期**；每次经过这四个阶段就会产生一**代**软件。除非项目“死亡”，否则通过重复同样的先启阶段、精化阶段、构建阶段和产品化阶段的顺序，产品将演进为下一代产品，但每

一次的侧重点都将放在不同的阶段上。这些随后的周期称为**演进周期。** 随着产品经历了几个周期，新一代产品随之产生。

1.2. 先启阶段

1.2.1. 目标

先启阶段的基本目标是实现项目的生命周期目标中所有相关因素（如客户等）之间的并行。 先启阶段主要对新的开发工作具有重大意义，新工作中的重要业务风险和需求风险问题必须在项目继续进行之前得到解决。对于重点是扩展现有系统的项目来说，先启阶段较短，但重点仍然是确保项目值得进行而且可以进行。

先启阶段的主要目标包括：

- 建立项目的软件规模和边界条件，包括运作前景、验收标准以及希望软件中包括和不包括的内容。
- 识别系统的关键用例（也就是将造成重要设计折衷操作的主要部分）。
- 评估整个项目的总体成本和进度（以及对即将进行的精化阶段进行更详细的评估）
- 评估潜在风险（不可预测性的来源）
- 准备项目的支持环境。

1.2.2. 核心活动

- **明确地说明项目规模**。这涉及了解环境以及最重要的需求和约束，以便于可以得出最终产品的验收标准。
- **计划和准备商业理由**。评估风险管理、人员配备、项目计划和成本/进度/收益率折衷的备选方案。
- **综合考虑备选构架**，评估设计和自制/外购/复用方面的折衷，从而估算出成本、进度和资源。此处的目标在于通过对一些概念的证实来证明可行性。该证明可采用可模拟需求的模型形式或用于探索被认为高风险区域的初始原型。先启阶段的原型设计工作应该限制在确信解决方案可行就可以了。该解决方案在精化和构建阶段实现。
- **准备项目的环境**，评估项目和组织，选择工具，决定流程中要改进的部分。

1.2.3. 里程碑：生命周期目标

生命周期目标里程碑评估项目的基本可行性。

先启阶段末是第一个重要的项目里程碑，即**生命周期目标里程碑**。此时，检查项目的生命周期目标，并决定继续进行项目还是取消项目。

1.2.3.1 评估标准

- 规模定义和成本 / 进度估算中，所有相关因素（如客户等）可并行
- 对是否已经获得正确的需求集达成一致意见，并且对这些需求的理解是共同的。
- 对成本 / 进度估算、优先级、风险和开发流程是否合适达成一致意见。
- 已经确定所有风险并且有针对每个风险的减轻风险策略。

如果项目无法达到该里程碑，则它可能中途失败或需要进行相当多的重新考虑。

1.2.3.2 提供的文档及模型

核心文档及模型（按照重要性排序）	里程碑状态
前景	已经对核心项目的需求、关键功能和主要约束进行了记录。
商业理由	已经确定并得到了批准。
风险列表	已经确定了最初的项目风险。
软件开发计划	已经确定了最初阶段及其持续时间和目标。软件开发计划中的资源估算（特别是时间、人员和开发环境成本）必须与商业理由一致。 资源估算可以涵盖整个项目直到交付所需的资源，也可以只包括进行精化阶段所需的资源。此时，整个项目所需的资源估算应该看作是大致的“粗略估计”。该估算在每个阶段和每次迭代中都会更新，并且随着每次迭代变得更加准确。

	根据项目的需要，可能在某种条件下完成了一个或多个附带的“计划”工件。此外，附带的“指南”工件通常也至少完成了“草稿”。
迭代计划	第一个精化迭代的迭代计划已经完成并经过了复审。
软件验收计划	完成复审并确定了基线；随着其他需求的发现，将对其在随后的迭代中进行改进。
项目专用模板	已使用文档模板制作了文档工件。
用例建模指南	确定了基线。
工具	选择了支持项目的所有工具。安装了对先启阶段的工作必要的工具。
词汇表	已经定义了重要的术语；完成了词汇表的复审。
用例模型（主角，用例）	已经确定了重要的主角和用例，只为最关键的用例简要说明了事件流。
领域模型（也叫做业务对象模型）	已经对系统中使用的核心概念进行了记录和复审。在核心概念之间存在特定关系的情况下，已用作对词汇表的补充。
原型	概念原型的一个或多个证据，以支持前景和商业理由、解决非常具体的风险。

1.3. 精化阶段

1.3.1. 目标

精化阶段的目标是建立系统构架的基线，以便为构建阶段的主要设计和实施工作提供一个稳定的基础。构架是基于对大多数重要需求（对系统构架有很大影响的需求）的考虑和风险评估发展而来的。构架的稳定性是通过一个或多个构架原型进行评估的。

精化阶段的主要目标包括：

- 确保构架、需求和计划足够稳定，充分减少风险，从而能够有预见性地确定完成开发所需的成本和进度。对大多数项目来说，通过此里程碑也就相当于从简单快速的低风险运作转移到高成本、高风险的运作，并且在组织结构方面面临许多不利因素。
- 处理在构架方面具有重要意义的所有项目风险
- 建立一个已确定基线的构架，它是通过处理构架方面重要的场景得到的，这些场景通常可以显示项目的最大技术风险。
- 制作产品质量构件的演进式原型，也可能同时制作一个或多个可放弃的探索性原型，以减小特定风险，例如：
  - 设计/需求折衷
  - 构件复用
  - 产品可行性或向客户和最终用户进行演示。
- 证明已建立基线的构架将在适当时间、以合理的成本支持系统需求。
- 建立支持环境。

为了实现这个主要目标，建立项目的支持环境也同等重要。这包括创建开发案例、创建模板和指南、安装工具。

1.3.2. 核心活动

- 快速**确定构架、确认构架并为构架建立基线**。
- 根据此阶段获得的新信息**改进前景**，对推动构架和计划决策的最关键用例建立可靠的了解。
- **为构建阶段创建详细的迭代计划并为其建立基线**。
- **改进开发案例，定位开发环境**，包括流程和支持构建团队所需的工具和自动化支持。
- **改进构架并选择构件**。 评估潜在构件，充分了解自制/外购/复用决策，以便有把握地确定构建阶段的成本和进度。集成了所选构架构件，并按主要场景进行了评估。通过这些活动得到的经验有可能导致重新设计构架、考虑替代设计或重新考虑需求。

1.3.3. 里程碑：生命周期构架

生命周期构架里程碑为系统构架建立管理基线，并使项目团队能够在构建阶段调整规模。

精化阶段末是第二个重要的项目里程碑，即**生命周期构架里程碑**。此时，您检查详细的系统目标和规模、选择的构架以及主要风险的解决方案。

1.3.3.1 评估标准

- 产品前景和需求是稳定的。

- 构架是稳定的。
- 可执行原型表明已经找到了主要的风险元素，并且得到妥善解决。
- 构建阶段的迭代计划足够详细和真实，可以保证工作继续进行。
- 构建阶段的迭代计划由可靠的估算支持。
- 所有客户方人员一致认为，如果在当前构架环境中执行当前计划来开发完整的系统，则当前的前景可以实现。
- 实际的资源耗费与计划的耗费相比是可以接受的。

如果项目无法达到该里程碑，则它可能中途失败或需要进行相当多的重新考虑。

1.3.3.2 提供的文档及模型

核心文档及模型（按照重要性排序）	里程碑状态
原型	已经创建了一个或多个可执行构架原型，以探索关键功能和构架上的重要场景。
风险列表	已经进行了更新和复审。新的风险可能是构架方面的，主要与处理非功能性需求有关。
项目专用模板	已使用文档模板制作了文档工件。
工具	已经安装了用于支持精化阶段工作的工具。
软件构架文档	编写完成并确定了基线，如果系统是分布式的或必须处理并行问题，则包括构架上重要用例的详细说明（用例视图）、关键机制和设计元素的标识（逻辑视图），以及（部署模型的）进程视图和部署视图的定义。
设计模型（和所有组成部分）	制作完成并确定了基线。已经定义了构架方面重要场景的用例实现，并将所需行为分配给了适当的设计元素。已经确定了构件并充分理解了自制/外购/复用决策，以便有把握地确定构建阶段的成本和进度。集成了所选构架构件，并按主要场景进行了评估。通过这些活动得到的经验有可能导致重新设计构架、考虑替代设计或重新考虑需求。
数据模型	制作完成并确定了基线。已经确定并复审了主要的数据模型元素（例如重要实体、关系和表）。

实施模型（以及所有组成工件，包括构件）	已经创建了最初结构，确定了主要构件并设计了原型。
前景	已经根据此阶段获得的新信息进行了改进，对推动构架和计划决策的最关键用例建立了可靠的了解。
软件开发计划	已经进行了更新和扩展，以便涵盖构建阶段和产品化阶段。
指南，如设计指南和编程指南。	使用指南对工作进行了支持。
迭代计划	已经完成并复审了构建阶段的迭代计划。
用例模型	用例模型（大约完成 80%）- 已经在用例模型调查中确定了所有用例、确定了所有主角并编写了**大部分**用例说明（需求分析）。
补充规约	已经对包括非功能性需求在内的补充需求进行了记录和复审。
可选	**里程碑状态**
商业理由	如果构架调查不涵盖变更基本项目假设的问题，则已经对商业理由进行了更新。
分析模型	可能作为正式工件进行了开发；进行了经常但不正式的维护，正演进为设计模型的早期版本。
培训材料	用户手册与其他培训材料。根据用例进行了初步起草。 如果系统具有复杂的用户界面，可能需要培训材料。

1.4. 构建阶段

1.4.1. 目标

构建阶段的目标是阐明剩余的需求，并基于已建立基线的构架完成系统开发。构建阶段从某种意义上来说是一个制造过程，在此过程中，重点在于管理资源和控制操作，以便优化成本、进度和质量。从这种意义上说，从先启和精化阶段到构建和产品化阶段，管理上的思维定势经历了从知识产权开发到可部署产品开发的转变。

构建阶段的主要目标包括：

- 通过优化资源和避免不必要的报废和返工，使开发成本降到最低。
- 快速达到足够好的质量
- 快速完成有用的版本（Alpha 版、Beta 版和其他测试发布版）
- 完成所有所需功能的分析、开发和测试。
- 迭代式、递增式地开发随时可以发布到用户群的完整产品。这意味着描述剩余的用例和其他需求，充实设计，完成实施，并测试软件。
- 确定软件、场地和用户是否已经为部署应用程序作好准备。
- 开发团队的工作实现某种程度的并行。 即使是较小的项目，也通常包括可以相互独立开发的构件，从而使各团队之间实现自然的并行（资源允许）。这种并行性可较大幅度地加速开发活动；但同时也增加了资源管理和工作流程同步的复杂程度。如果要实现任何重要的并行，强壮的构架至关重要。

1.4.2. 核心活动

- 资源管理，控制和流程优化
- 完成构件开发并根据已定义的评估标准进行测试
- 根据前景的验收标准对产品发布版进行评估。

1.4.3. 里程碑：最初操作性能

最初操作性能里程碑确定产品是否已经可以部署到 Beta 测试环境。

在最初操作性能里程碑，产品随时可以移交给产品化团队。此时，已开发了所有功能，并完成了所有 Alpha 测试（如果有测试）。除了软件之外，用户手册也已经完成，而且有对当前发布版的说明。

1.4.3.1 评估标准

构建阶段的评估标准涉及到对以下问题的回答：

- 该产品发布版是否足够稳定和成熟，可部署在用户群中？
- 是否已准备好将产品发布到用户群？
- 实际的资源耗费与计划的相比是否仍可以接受？

如果项目无法达到该里程碑，产品化可能要推迟一个发布版。

1.4.3.2 提供的文档及模型

核心文档及模型（按照重要性排序）	里程碑状态
“系统”	可执行系统本身随时可以进行“Beta”测试。
部署计划	已开发最初版本、进行了复审并建立了基线。
实施模型（以及所有组成部分，包括构件）	对在精化阶段创建的模型进行了扩展；构建阶段末期完成所有构件的创建。
测试模型（和所有组成部分）	为验证构建阶段所创建的可执行发布版而设计并开发的测试。
培训材料	用户手册与其他培训材料。根据用例进行了初步起草。如果系统具有复杂的用户界面，可能需要培训材料。
迭代计划	已经完成并复审了产品化阶段的迭代计划。
设计模型（和所有组成部分）	已经用新设计元素进行了更新，这些设计元素是在完成所有需求期间确定的。
项目专用模板	已使用文档模板制作了文档模板。
工具	已经安装了用于支持构建阶段工作的工具。
数据模型	已经用支持持续实施所需的所有元素（例如，表、索引、对象关系型映射等）进行了更新
可选	**里程碑状态**
补充规约	已经用构建阶段发现的新需求（如果有）进行了更新。
用例模型（主角，用例）	已经用构建阶段发现的新用例（如果有）进行了更新。

1.5. 产品化阶段

1.5.1. 目标

产品化阶段的重点是确保最终用户可以使用软件。产品化阶段可跨越几个迭代，包括测试处于发布准备中的产品和基于用户反馈进行较小的调整。在生命周期中的该点处，用户反馈应主要侧重于调整产品、配置、安装和可用性问题，所有较大的结构上的问题应该在项目生命周期的早期阶段就已得到解决。

在产品化阶段生命周期结束时，目标应该已经实现，项目应处于将结束的状态。某些情况下，当前生命周期的结束可能是同一产品另一生命周期的开始，从而导致产生产品的下一代或下一版本。对于其他项目，产品化阶段结束时可能就将文档与模型完全交付给第三方，第三方负责已交付系统的操作、维护和扩展。

根据产品的种类，产品化阶段可能非常简单，也可能非常复杂。例如，发布现有桌面产品的新发布版可能十分简单，而替换一个国家的航空交通管制系统可能就非常复杂。

产品化阶段的迭代期间所进行的活动取决于目标。例如，在进行调试时，实施和测试通常就足够了。但是，如果要添加新功能，迭代类似于构建阶段中的迭代，需要进行分析设计。

当基线已经足够完善，可以部署到最终用户领域中时，则进入产品化阶段。通常，这要求系统的某个可用部分已经达到了可接受的质量级别并完成用户文档，从而向用户的转移可以为所有方面都带来积极的结果。

产品化阶段的主要目标是：

- 进行 Beta 测试，按用户的期望确认新系统
- Beta 测试和相对于正在替换的遗留系统的并行操作
- 转换操作数据库
- 培训用户和维护人员
- 市场营销、进行分发和向销售人员进行新产品介绍
- 与部署相关的工程，例如接入、商业包装和生产、销售介绍、现场人员培训
- 调整活动，如进行调试、性能或可用性的增强
- 根据产品的完整前景和验收标准，对部署基线进行的评估
- 实现用户的自我支持能力
- 在用户之间达成共识，即部署基线已完成

- 在用户之间达成共识，即部署基线与前景的评估标准一致

1.5.2. 核心活动

- 执行部署计划
- 对最终用户支持材料定稿
- 在开发现场测试可交付产品
- 制作产品发布版
- 获得用户反馈
- 基于反馈调整产品
- 使最终用户可以使用产品

1.5.3. 里程碑：产品发布

产品化阶段末是第四个重要的项目里程碑，即**产品发布里程碑**。此时，您确定是否达到目标，以及是否应该开始另一个开发周期。有时候，该里程碑可能与下一周期的先启阶段末重合。产品发布里程碑是项目验收复审成功完成的结果。

1.5.3.1 评估标准

产品化阶段的主要评估标准涉及到对以下问题的回答：

- 用户是否满意？
- 实际的资源耗费与计划的耗费相比是否可以接受？

在产品发布里程碑处，发布后的维护周期同时开始。这涉及开始一个新的周期，或某个其他的维护发布版。

1.5.3.2 提供的文档及模型

核心文档及模型（按照重要性排序）	里程碑状态
产品工作版本	已按照产品需求完成。客户应该可以使用最终产品。

发布说明	完成。
安装产品与模型	完成。
培训材料	完成，以确保客户自己可以使用和维护产品。
最终用户支持材料	完成，以确保客户自己可以使用和维护产品。
可选	**里程碑状态**
测试模型	在客户想要进行现场测试的情况下，可以提供测试模型。

※※※

成功项目管理的秘密

讨论

摘自 SDMagazine，Karl Wiegers 著，Shids 译

在最好的情况下，管理软件项目也是很困难的。不幸的是，许多新项目经理实质上没有受到任何就职培训。这里有 20 个成功的管理经验供项目经理参考。

1. 定义项目成功的标准

在项目的开始，要保证风险承担者对于他们如何判断项目是否成功有统一的认识。经常，满足一个预定义的进度安排是唯一明显的成功因素，但是肯定还有其他的因素存在，比如：增加市场占有率，获得指定的销售量或销售额，取得特定用户满意程度，淘汰一个高维护需求的遗留系统，取得一个特定的事务处理量并保证正确性。

2. 识别项目的驱动、约束和自由程度

每个项目都需要平衡它的功能性，人员，预算，进度和质量目标。我们把以上五个项目方面中的每一个方面，要么定义成一个约束，你必须在这个约束中进行操作，要么定义成与项目成功对应的驱动，或者定义成通向成功的自由程度，你可以在一个规定的范围内调整。相关的详细信息，请参照我的《创建一种软件工程文化》(Creating a Software Engineering Culture)(Dorset House, 1996)中的第二章。

3. 定义产品发布标准

在项目早期，要决定用什么标准来确定产品是否准备好发布了。你可以把发布标准基于：还存在有多少个高优先级的缺陷，性能度量，特定功能完全可操作，或其他方面表明项目已经达到了它的目的。不管你选择了什么标准，都应该是可实现的、可测量的、文档化的，并且与你的客户指的“质量”一致。

4. 沟通承诺

尽管有承诺不可能事件的压力，从不作一个你知道你不能保证的承诺。和客户和管理人员沟通哪些可以实际取得时，要有好的信誉。你的任何以前项目的数据会帮助你作说服的论据，虽然这对于不讲道理的人来说没有任何真正的防御作用。

5. 写一个计划

有些人认为，花时间写计划还不如花时间写代码，但是我不这么认为。困难的部分不是写计划。困难

的部分是作这个计划--思考，沟通，权衡，交流，提问并且倾听。你用来分析解决问题需要花费的时间，会减少项目以后会带给你的意外。

6. 把任务分解成英寸大小的小圆石

英寸大小的小圆石是缩小了的里程碑。把大任务分解成多个小任务，帮助你更加精确的估计它们，暴露出在其他情况下你可能没有想到的工作活动，并且保证更加精确、细密的状态跟踪。

7. 为通用的大任务开发计划工作表

如果你的组经常承担某种特定的通用任务，如实现一个新的对象类，你需要为这些任务开发一个活动检查列表和计划工作表。每个检查列表应该包括这个大任务可能需要的所有步骤。这些检查列表和工作表将帮助小组成员确定和评估与他/她必须处理的大任务的每个实例相关的工作量。

8. 计划中，在质量控制活动后应该有修改工作

几乎所有的质量控制活动，如测试和技术评审，都会发现缺陷或其他提高的可能。你的项目进度或工作细分结构，应该把每次质量控制活动后的修改，作为一个单独的任务包括进去。如果你事实上不用作任何的修改，很好，你已经走在了本任务的计划前面。但是不要去指望它。

9. 为过程改进安排时间

你的小组成员已经淹没在他们当前的项目中，但是如果你想把你的组提升到一个更高的软件工程能力水平，你就必须投资一些时间在过程改进上。从你的项目进度中留出一些时间，因为软件项目活动应该包括做能够帮助你下一个项目更加成功的过程改进。不要把你项目成员可以利用的时间 100%的投入到项目任务中，然后惊讶于为什么他们在主动提高方面没有任何进展。

10. 管理项目的风险

如果你不去识别和控制风险，那么它们会控制你。在项目计划时花一些时间集体讨论可能的风险因素，评估它们的潜在危害，并且决定你如何减轻或预防它们。要一个软件风险管理的简要的指南，参见我的文章“Know Your Enemy: Software Risk Management”（Oct. 1998）。

11. 根据工作计划而不是日历来作估计

人们通常以日历时间作估计，但是我倾向于估计与任务相关联的工作计划（以人时为单位）的数量，然后把工作计划转换为日历时间的估计。这个转换基于每天我有多少有效的小时花费在项目任务上，我可能碰到的任何打断或突发调整请求，会议，和所有其他会让时间消失的地方。

12. 不要为人员安排超过他们 80%的时间

跟踪你的组员每周实际花费在项目指定工作的平均小时数，实在会让人吃惊。与我们被要求做的许多活动相关的任务切换的开销，显著地降低了我们的工作效率。不要只是因为有人在一项特定工作上每周花费 10 小时，就去假设他或她可以马上做 4 个这种任务，如果他或她能够处理完 3 个任务，你就很幸运了。

13. 将培训时间放到计划中

确定你的组员每年在培训上花费多少时间，并把它从组员工作在指定项目任务上的可用时间中减去。你可能在平均值中早已经减去了休假时间、生病时间和其他的时间，对于培训时间也要同样的处理。

14. 记录你的估算和你是如何达到估算的

当你准备估算你的工作时，把它们记录下来，并且记录你是如何完成每个任务的。理解创建估算所用的假设和方法，能够使它们在必要的时候更容易防护和调整，而且它将帮助你改善你的估算过程。

15. 记录估算并且使用估算工具

有很多商业工具可以帮助你估算整个项目。根据它们真实项目经验的巨大数据库，这些工具可以给你一个可能的进度和人员分配安排选择。它们同样能够帮助你避免进入“不可能区域”，即产品大小，小组大小和进度安排组合起来没有已知项目成功的情况。Software Productivity Centre(www.spc.ca)公司的 Estimate Pro 是可以一试的好工具。

16. 遵守学习曲线

如果你在项目中第一次尝试新的过程，工具或技术，你必须认可付出短期内生产力降低的代价。不要期望在新软件工程方法的第一次尝试中就获得惊人的效益，在进度安排中考虑不可避免的学习曲线。

17. 考虑意外缓冲

事情不会象你项目计划的一样准确的进行，所以你的预算和进度安排应该在主要阶段后面包括一些意外的缓冲，以适应无法预料的事件。不幸的是，你的管理者或客户可能把这些缓冲作为填料，而不是明智的承认事实确实如此。指明一些以前项目不愉快的意外，来说明你的深谋远虑。

18. 记录实际情况与估算情况

如果你不记录花费在每项任务上的实际工作时间，并和你的估算作比较，你将永远不能提高你的估算能力。你的估算将永远是猜测。

19. 只有当任务 100%完成时，才认为该任务完成

使用英寸大小的小圆石的一个好处是，你可以区分每个小任务要么完成了，要么没有完成，这比估计一个大任务在某个时候完成了多少百分比要实在的多。不要让人们只入不舍他们任务的完成状态；使用明

确的标准来判断一个步骤是否真正的完成了。

20. 公开、公正地跟踪项目状态

创建一个良好的风气，让项目成员对准确地报告项目的状态感到安全。努力让项目在准确的、基于数据的事实基础上运行，而不是从因为害怕报告坏消息而产生的令人误解的乐观主义。使用项目状态信息在必要的时候进行纠正操作，并且在条件允许时进行表扬。

这些提示不能保证你的成功，但是它们将帮助你在你的项目上获得一个坚实的把手，并且保证你做了所有你可以做的事来让项目在这个疯狂的世界上成功。

原文链接：http://www.processimpact.com/articles/proj_mgmt_tips.html

UML 相关产品价格

讨论

摘自 ObjectsByDesign，Cliff(cliff@szonline.net)译

最后更新：5/15/2001

公司	产品	版本	日期	平台	价格
Adaptive Arts	Simply Objects 标准版	3.2	Mar-01	Windows	**$269**
	面向 Delphi, Smalltalk, Eiffel, Java, C++, Corba, VB 的正向工程工具 ， 以 jpeg,png 格式输出图表				
Adaptive Arts	Simply Objects 专业版	3.2	Mar-01	Windows	**$1999**
	比标准版多了 Use Case 和交互图表（interaction diagrams），报表生成器，多用户等功能				
Aonix	StP/UML	7.1	Jan-99	Windows, Unix	**n/a**
	多用户库，综合了 DOORS，面向 Forte, Smalltalk, Java, C++ 等				
Arion Software	UML2COM	1	Feb-01	Windows	**$990**
	面向 VC++/C++的综合工具，增添了对 Rational Rose 的支持和 COM+代码生成器				
Artisan	Real-time Modeler	4	Feb-01	Windows	**$2495**
	实时建模，多用户对象库				
Artisan	Real-time Studio 专业版	4	Feb-01	Windows	**n/a**
	面向 C, C++, Java 的双向工程工具 综合了 DOORS，支持状态机动画				
Atos Origin	Delphia Object Modeler (D.OM)	3.2.6	Dec-00	Windows	**$0**
	从 UML 模型自动生成功能模型，支持 XMI、类和状态图表、报表生成				
BoldSoft	Bold for Delphi	3	Mar-01	Delphi	**$2900**
	OCL，面向 Delphi, SQL 的正向工程工具				
Computer Associates	Paradigm Plus	3.7	Sep-98	Windows, Unix	**n/a**
	多重代码生成，对象数据库，综合了数据建模技术				
Confluent	Visual Thought	1.4		Windows, Unix	**$695**
	支持多平台图表和浮动图表工具				
Dia	Dia	0.86	Aug-00	Linux	**$0**

	Gnome 下的类似 Visio 的图表工具，使用了一个 UML 模板，以 SVG 格式输出				
Documentator	Documentator	2.9.8	Feb-01	Windows	**$0**
	将 Rose 2000 的文档转换成 MS Word 格式				
Elixir Technologies	Elixer CASE	1.2.4	Nov-99	Java VM	**$295**
	自动产生序列图表，支持 OCL, XMI				
Embarcadero Technologies	GDPro (Windows)	5	Oct-00	Windows	**$2995**
	面向 Java、C++的双向工程工具，支持 EJB				
Embarcadero Technologies	GDPro (Unix)	5	Oct-00	Unix	**$4795**
	面向 Java、C++的双向工程工具，支持 EJB				
Excel Software	Win A&D 标准版	3.1	Dec-99	Windows	**$495**
	支持 CRC 卡，构件模型				
Excel Software	Win A&D 桌面版	3.1	Dec-99	Windows	**$1295**
	增添对 Java, Delphi, C++, 脚本，状态模型的支持				
Honeywell	DOME	5.3	Mar-00	Smalltalk	**$0**
	扩展符号集，符合 GNU GPL 协议，用 Smalltalk 开发 有专门 FTP 站点交换工程模型				
Hoora	HAT 专业版	3.1	Mar-01	Windows	**$800**
	支持 HOORA 进程，面向 C++ 的正向工程工具，有报表生成、需求管理、导入 Rose 文件等功能				
I-Logix	Rhapsody Modeler	2.3	Sep-00	Windows	**$0**
	实时工具、面向 C, C++，单用户，免费的初学者版本				
I-Logix	Rhapsody Solo	2.3	Sep-00	Windows	**$895**
	实时工具，面向 C, C++, Java，单用户，支持 XMI				
I-Logix	Rhapsody Development	2.3	Sep-00	Windows	**n/a**
	实时工具，面向 C, C++, Java，多用户，支持 XMI				
IBM	Visual Age Smalltalk UML Designer	5	Mar-00	Smalltalk	**$3419**
	Visual Age for Smalltalk 企业版的 UML 设计器				
Innovative Software	Object Engineering Workbench	3	Nov-98	Windows	**$490**
	面向 C++, Java; Poet OODBMS 库				
Kennedy-Carter	iUML	2	Jan-01	Windows, Unix	**$3000**
	使用一种行为语言产生可执行的 UML 模型，包括代码生成器和模拟工具				

Mega International	Mega Suite			Windows	**n/a**
	支持 Delphi, Forte, Java, VB				
Mesa	MesaVista for Rational Rose			Java VM	**n/a**
	访问 Rose 模型的网页浏览器				
MetaCase Consulting	MetaEdit+	3		Windows, Unix	**$4500**
	多用户对象库，可自定义的 meta 工具，报表生成，面向 Java, C++, Smalltalk, IDL, Delphi, SQL				
Metamill Software	Metamill	1	May-01	Windows	**$75**
	廉价的工具，在共享库的基础上对文件进行索引。Java 和 C++代码生成。				
MicroTOOL	ObjectiF	4.5		Windows	**n/a**
	面向 VB 脚本, C++, Java, IDL, 可与 JBuilder 结合在一起				
Microgold Software	WithClass 专业版	2000	Sep-00	Windows	**$495**
	多重代码生成，面向 Python 脚本，现已支持 C#				
Microgold Software	WithClass 企业版	2000	Sep-00	Windows	**$800**
	比专业版多对 VBA 的支持				
Microsoft	Visual Modeler	2		Windows	**n/a**
	Rational Rose 面向 VS 6.0 版本的一个子集				
Microsoft Visio	Visio 2000 专业版	2000	Jan-00	Windows	**$399**
	C++, VB 的逆向工程工具，支持 MS Visual Studio				
Microsoft Visio	Visio 2000 企业版	2000	Jan-00	Windows	**$999**
	比专业版多了面向 C++, VB 的正向工程功能，包括了 MS Repository 2.0, 支持报表生成				
Mid	Innovator CASE	6.2		Windows, Unix	**n/a**
	在线资料库，支持 Forte, Smalltalk, Java, C++ 数据和商业进程模型				
ModelMaker Tools	ModelMaker	5.4	Mar-01	Delphi	**$249**
	面向 Delphi 的正向和逆向工程工具，支持 GOF 设计模式				
Modelistic	Modelistic	1	Aug-00	Java VM	**$299**
	面向 Java 的双向工程工具				
Mountfield Computers	mUML	2.2.3	May-01	Java VM	**$0**
	支持 UML 1.3 的所有 9 种图表，非商业用途可自由使用，面向 Java 的双向工程工具				
No Magic	MagicDraw UML 标准版	4.5	May-01	Java VM	**$249**
	支持 UML1.3 的所有 9 种图表，支持 Swing 用户界面，HTML 生成，可读				

	入 Rose 模型，支持 XMI, SVG, XSLT				
No Magic	MagicDraw UML 专业版	4.5	May-01	Java VM	**$499**
	比标准版多了面向 Java、C++的正向和逆向工程功能，支持 IDL				
Novosoft	Novosoft UML Library	0.4.19	Feb-01	Java VM	**$0**
	支持 UML1.3 模型的开放式资源库，坚持使用 XMI，并结合 ArgoUML 技术				
Novosoft	FL		Oct-00	Java VM	**n/a**
	从类图表开发 Java 对象，Rational Rose 插件,支持 OQL，DBMS				
OTW Software	Object Technology Workbench 个人版	2.4	Apr-00	Windows	**$495**
	面向 Java 和 C++的双向工程工具，支持 CORBA-IDL, SQL-DDL, 模式，对象库和 HTML				
OTW Software	Object Technology Workbench 团队版	2.4	Apr-00	Windows	**$995**
	增加了团队共享库功能				
OWiS Software (Germany)	OTW - Object Technology Workbench	2.4	Jan-00	Windows	**n/a**
	面向 Java 和 C++的双向工程工具，支持 CORBA-IDL, SQL-DDL, 模式，对象库和数据模型				
Object Domain Systems	Object Domain 标准版	2.5	Jun-99	Java VM	**$495**
	支持 C++, Java, Python, Python 脚本的正向和逆向工程工具，价格适合作教学用途				
Object Domain Systems	Object Domain 专业版	2.5	Jun-99	Java VM	**$995**
	比标准版多增多用户库以及面向 Java 的双向工程，HTML 生成等功能				
Object Insight	JVISION	1.4.2	Nov-00	Java VM	**$399**
	面向 Java 的逆向工程工具，与 Visual Cafe 结合使用，支持 HTML 生成				
Object Plant	Object Plant	2.1.7	Mar-00	MacOS	**$35**
	共享软件，支持类、状态和 UseCase 图表，面向 C++, Java				
Plastic Software	Plastic	3	Jan-01	Java VM	**$297**
	面向 Java 的双向工程工具，支持 HTML 生成，模型验证				
Popkin	System Architect	2001	Mar-00	Windows	**n/a**
	面向 Java, C++, VBA 的双向工程工具，支持数据模型，支持微软 Microsoft repository 软件，脚本等				
Pragsoft Corporation	UML Studio	5	Dec-00	Windows	**$500**
	面向 C++, Java 的正向和逆向工程工具，支持 IDL，脚本工具，自动保存				

Princeton Softech	Select/Enterprise	6		Windows	**n/a**
	构件仓库，结合数据建模，面向 C++, Java, Forte, VB 的双向工程工具				
ProxySource	ProxyDesigner	1	Dec-00	Windows	**$0**
	向在线论坛直接发布模型				
Ptech Inc.	FrameWork	5.4	Sep-99	Windows	**n/a**
	企业和进程模型，商业分析工具				
Rational	Rose Modeler	2001	Nov-00	Windows	**$1829**
	基本模型工具				
Rational	Rose 专业版	2001	Nov-00	Windows	**$2442**
	比普通版多了双向工程功能，支持共享库，数据建模，Java, C++, VB 版本单独销售				
Rational	Rose 企业版	2001	Nov-00	Windows	**$4290**
	比专业版多了网页发布功能，支持 CORBA-IDL, 与 ClearCase (版本控制) and MS Visual Studio (只包括 VB, C++) 结合使用				
Rational	Rose Real Time	2001	Nov-00	Windows	**n/a**
	基于 ObjectTime 技术的实时建模工具				
Softeam	Objecteering 个人版	4.3.2	Jan-01	Windows, Unix	**$0**
	免费的基础版本，包括了对 XMI 的支持				
Softeam	Objecteering 个人版/ Java	4.3.2	Jan-01	Windows, Unix	**$859**
	多了对 Java 的双向工程功能				
Softeam	Objecteering 工程版	4.3.2	Jan-01	Windows, Unix	**$1569**
	除了多用户共享库之外包括其他所有功能的版本				
Softeam	Objecteering 企业版	4.3.2	Jan-01	Windows, Unix	**$2569**
	比工程版多了参数化的代码生成功能、多用户共享库、数据建模、设计模式等				
Softera	SoftModeler 标准版	3	Apr-01	Java VM	**$245**
	基础建模				
Softera	SoftModeler 专业版	3	Apr-01	Java VM	**$495**
	支持 EJB，双向同步				
Softera	SoftModeler 企业版	3	Apr-01	Java VM	**$995**
	比专业版多了多用户支持、共享库、模型模拟等功能				
Sparx Systems	Enterprise Architect	2.1	Apr-01	Windows	**$149**

	专业版				
	面向 C++、Java、VB 的双向工程工具，支持 Use Cases，多用户，工程评估等，并带有一个优秀的免费 UML 指导				
Stingray/RogueWave	UML Studio		Feb-00	Windows	**$495**
	面向 Visual C++的建模工具，与 MS VisualStudio 一起使用，可下载一个关于黄貂鱼产品的 UML 模型				
Sybase	PowerDesigner	8	Mar-01	Windows	**$5990**
	使用类图进行对象/关系设计，支持共享库，包括 Use Case 和序列图表				
TNI	OpenTool	3.2	Jan-01	Windows, Unix	**n/a**
	面向 C++, Java, Smalltalk 的正向工程工具，可对 Java 进行逆向工程，提供多种文档生成				
Telelogic	ObjectGeode	4.1	Jun-99	Windows, Unix	**n/a**
	实时建模，多个 RTOS 目标，生成 C, C++ 代码				
Telelogic	Tau UML Suite	4.2	Mar-01	Windows	**n/a**
	实时建模，支持 UML 到 SDL 的转换,支持 XMI 已从 Sterling Software 得到 COOL:Jex 技术，从 QSS 得到 DOORS 技术				
The Object Factory	Optimize	4.1	Mar-01	Windows	**$1995**
	基于 OO 标准的工程评估和计划工具，提供与 Rational Rose, Select Enterprise, Artison Real-Time Studio 的接口				
Tigris	ArgoUML	0.81	Oct-00	Java VM	**$0**
	开放源码工程，用 Java 开发，支持运行时可模型评估，支持 OCL, XMI				
TogetherSoft	Together 公众版	5	May-01	Java VM	**$0**
	面向 Java,C++的同时双向工程工具，只支持类图				
TogetherSoft	Together 个人版	5	May-01	Java VM	**$3495**
	比公众版多了 UML 图表支持，HTML 生成，源码调试器等				
TogetherSoft	Together 控制中心	5	May-01	Java VM	**$5995**
	比个人版多了 EJB 开发和发布支持，支持 GOF 设计模式				
Unimodeler	Unimodeler	0.3	Jan-00	Linux	**$0**
	支持所有 9 种 UML 图表，基于 GTK (Gnome) ，支持脚本打印				
Visual Object Modelers	Visual UML 标准版	2.7.1	Mar-01	Windows	**$495**
	支持 VBScript，提供 OLE 自动控制服务器				
Visual Object Modelers	Visual UML 标准版 for VB	2.7.1	Mar-01	Windows	**$795**
	面向 VB 提供双向工程功能				
Visual Object Modelers	Visual UML 加强版 for VB	2.7.1	Mar-01	Windows	**$995**

	包括 VBA 支持，包括 MS Repository 2.0				
WebGain	StructureBuilder 专家版	4	Sep-00	Java VM	**$999**
	增添了 sequence 和 use-case 图表支持，开放时 API，HTML 生成				
WebGain	StructureBuilder 企业版	4	Sep-00	Java VM	**$1995**
	增添了 EJB 支持， sequence 图表的双向工程功能(独一无二的功能!)				
Xaan's Lair	Thorn	0.2.11	Feb-01	Java VM	**$0**
	开放源码的 Java UML 工具，支持 HTML 生成，XML 模型支持				

论坛-已有 7000 多人的讨论组>>

用 use case 驱动 ooa,ood,oop 的困惑…

ooandoo:

根据 rup，用 usecase 驱动 ooa,ood,oop，即对每一个 usecase，从分析设计一直到产品，然后开始第二个 usecase...，再开始第三个 usecase...。我们知道，这些 usecase 中有很多 class 是共享的。这样就导致了问题：做第二个 usecase 时必将对第一个 usecase 中的一些共享的 class 进行修改，即后面做的 usecase 要对前面做的进行修改。这些修改有可能使前面 usecase 驱动的产品毁坏。

王正光:

1,这个问题很好。
2,面向对象设计和编程的重要方面是代码重用。
3,在多个 USECASE 中，我们知道好多类是可以重用的，但可能接口（INTERFACE）不一样，否则的话，USECASE 中用的是同一个类。
4,所以我们可以设计具有多个接口（INTERFACE）的类，实现类在多个 USECASE 中的重用。

Jasperyeh:

按照我的理解，我首先讲述“固化”的概念：对于 class(?)，引用继承机制，我们可以产生基类的变体。这个子类中的接口允许一定程度上的不同，当然不是本质的不同。

实际上，这是一个固化的过程。我所谓“固化”，是指基类的定义和接口，在一定的进度之后，应该予以确定。这样，假定以后你需要一个略微不同的 class，你需要从固化类中继承一个中间类来实现要求。

除了对象理论的参照之外，明眼人一见便知，这和微软的所谓“界面固化”（UI Frozen）的道理有相似之处。推而广之，算法固化，设计方案固化，...，（我想的太乱了）。

其次，更新与自适应：

类的集成和类层次树的形成，一个目的就是自动得到新特性。假定 A 类接口形成，B 继承于 A。当 A 暴露的接口改变时，B 将会自动获得 A 的改变，这是强大的能力！应该善用之。

请注意这和“固化”没有任何冲突：改动 A 的暴露接口时，第一原则是已经暴露的接口不应该被修改。

这里体现一个软件开发人员的分析功力了，你是否能够让自己以前设定的接口一直适用下去呢？如果不能，应该继续加强分析(即对对象进行抽象)能力，不再赘言。要言之，你可以对暴露接口进行添加，不可以改动和删除，这将导致雪崩的不可测后果。

第二原则是，隐藏的属性和内部的接口允许被有节制的调整，将不会影响到继承类。

同样，这也是一种能力，应该掌握那些可以被暴露，哪些不应该被暴露给继承类。关于这一点，想想 Undocumented Windows API 吧。要想一棵树有序，你需要在设计时做出 n 个决定，我相信，每一个决定都是“痛苦”的。

而已经做好的类设计只能有节制的被调整，盖因如果你改了个底朝天的话，岂不是说你以前的设计太差，只能换掉？

最后，版本控制：也许我们没有必要持续保持接口的一致性，你当然可以这么做，而我也当然不会反对。那么，请做好相应的说明文档，标记上合适的版本跟踪。 别忘了，并不是只有源代码的，文档，设计稿，图样，任何东西都可以进行版本跟踪。

以后，不知道，我会不会被 check out 为 jy77.0.20500107，;)

一家之言，你可以参考并调整你的类和类图设计。并，敬请指正。

如何确定 actor？

Charleysong：

假设有这样一个系统：包括发送程序和接收程序，发送程序运行在北京的一台计算机上，由甲操作员操作；接收程序运行在上海的一台计算机上，由乙操作员操作；发送程序拨号连接到上海的计算机，并向接收程序传送数据。我认为甲、乙操作员理所当然是 actor，那么，发送程序和接收程序是不是 actor 呢？为什么？请各位多多指教。谢谢。

liaofan：

不是，actor 是系统外界与系统交互的，发送和接收程序属于系统内部，所以不是。

Caidehui：

确定系统的范围，寻找谁使用系统（角色），确定角色的目的（用例）

我觉得要区分角色、用例，首先要确定系统的范围，如果我们是要建立一个远程的文件发送与接受系统，那就考虑谁会使用系统，那就是甲和乙。

现在考虑，甲使用系统的目的是什么，是发送文件给乙，用例就是发送文件。乙使用系统的目的是什么，那就是从甲接收文件，它的用例就是接收文件。

关于 Mouri 提出的有关自动程序的问题，只是实现上的事，角色还是两个，用例也是两个。当然用例的大小也是需要考虑的问题，这里这两个用例好像联系非常紧密，为此我们需要考虑是不是实际上是一个用例：远程文件传递。从甲到乙传递建立顺序模型，两者之间传递应该遵循的协议在图上能够很好的体现。

类之间设置成“关联”or“依赖”似乎全在个人喜好…

ooandoo：

类之间的联系设置成“关联(association)”or“依赖(dependency)”似乎全在个人喜好，有无一个容易操作的判断标准？

Jasperyeh：

一个项目，目的是建立文章库，其中要做的是收集各种书籍，文章，期刊，杂志，等等文字类的素材。

于是，基于此，评论文章和书，是关联关系。评论文章可以单独被收列在文章库目录列表中。

如果，第二个项目，我们要建立的是书库，目的是收集古今中外的著名书籍，长篇的，短篇的，古代的，现代的。

那么，评论文章和书之间，就存在依赖关系，盖因这时候，评论文章不能独立成为一个目标对象了。它不能被单独收列在目录列表中。

唉，这个例子还不太准确，算了，回头有空我再举个象话一点的吧，:)

jasper

mouri：

我的直观看法：书本身是客观存在的，有没有相应的评论文章，它都是一本书。而评论文章就不同了，如果没有了评论对象，它也就没有存在的价值了。这决不是个人喜好可以改变的关系。

Jasperyeh：

但我要想强调的是区别，而不是 reality 中的存在关系。

在具体的项目中，对象已经存在了，这时候如何对待对象，取决于我们要干什么，目的不同，可能导致对象间的关系不同。即使是同样两个对象，因为利用这俩对象完成的目标不同，也导致我们对其关系的设定有所不同。

（应该看到 OO 的目标固然是为了准确反应 real world，但是 OO 的更大目标是为了提供一种描述方式来描述现实，以便以一种简便直观的方法来处理它。）

所以，我举那样一个例子，尽管不太合适。也许，下一次，我还是设法从图形处理上弄个例子来吧。

Zhushuangmin：

"关联"和"依赖"是最容易区分的.把"关联"和"组合","聚合"搞混到是有,但很少有人把"关联"和"依赖"搞混的.

关联是类之间的联系,是客观存在的,是在某活动中产生的,例如学生和老师,因为学生要听老师上课,所以学生和老师之间就有关联,而依赖是一种引用关系, 例如类中的某个方法的参数的类型是某个类,或方法种引用了某类的对象等.

不知这样解释行不?,可以讨论一下

jackmoongong：

好象不是这样吧，关联里经常会有引用的。就如组合就标记为 aggrecation by reference(or unspecifed)，而聚合就为 aggrecattion by value。我个人认为组合，聚合是一种强制程度的关系。如果你采用聚合，而散失了必要的灵活性，就采用组合，采用组合仍然不能解决其它问题就采用一般关联。

Kinsing：

Some notations of UML is similar with the notation of Coad/Yordon , so if you once read the books of OOA/OOD written by Coad/Youdon , you can grasp the the notation of UML through comparing them to them of OOA/OOD .

UML Coad/Youdon

1 Association(UML) : instance connection (Coad/Youdon)+ Message link(Coad/Youdon)

2 Aggregation(UML) : whole/part structure (Coad/Youdon)

3 dependency(UML) : message link without association

for example :

A: the type of one parameter of a service uses the supplier class

B: the return value of a service uses the supplier class

Xingzeng：

区分“关联”or“依赖”必须根据你的 requirement

In one situation you may have two classes in association, and in other situation you may have them in dependency. This purely depends on your requirement. Normally you can based on the following rule to set the relation:

Assume you have two class A and B, if you set A and B in association, you can get A instance from B instance or get B instance from A instance depending the direction of the association; if you only set dependency relation, normally you do not have the capability to get the one class instance from other one.

Woodysteven：

让我来把它说得清楚点：

关联本身即是一种依赖。共同点是：

1，一个类依赖另一个类，意味着依赖的类在源代码级别引用了被依赖的类，你需要 #include(c++)或 import

2，如果被依赖的类的接口发生了改变，将导致依赖它的类重新编译。

不同点是：关联是一种更特殊的依赖。表现

为：关联的类“静态”地引用了被关联类的实例变量，这种引用通常表现为被关联类以类属性的形式出现在关联类的类定义中，也可以表现为关联类引用了一个类型为被关联类的全局变量。相反，依赖却是比较动态的，通常表现为依赖的类的某个方法以被依赖的类作为其参数。依赖还有一种形式是：若 class A 的某个方法创造了 class B 的实例，则 class A 依赖于 class B。

若 A 既依赖 B 又和 B 关联，则它们是关联关系，由此可见，关联比依赖更强。

Rose 等其他 Case 工具使用的疑问！请指教！

an_xin：

无论采取什么养的 case 工具，在我看来总是有些疑问。

1. 数据源从哪里来？

从需求分析来 ？系统设计来？还是从 OOA 来？还是 OOD?还是替代了需求分析？如果不能够替代需求分析，系统设计？那么这些工具不就成了画图工具了？Visio 2002 能够算得上 Case 工具么？Case 的作用到底是什么？

2. 数据到哪里去？

画出了系统的 Use Case 图，结构图，流程图等等。说到底还是为了实现/编程，这些图形能够很好的表示出软件的体系结构么？是不是能够方便程序员编写程序时候对系统模块的理解呢？

产生以上疑问的原因是很多人认为我使用 CASE 工具是在增加自己的工作量，那些图形出来还不如文字描述更让程序员看得懂。请高手表自己的看法。

Piaoyun：

我乡下舅舅有 7 个儿子，老大是省城里的工程师，建房子就知道画图，有时几个晚上能赚数千上万元，却从来不知道和泥水、砌砖。可房子建得再好再漂亮人们总是跨他，那迎着烈日顶着风雨的泥水匠，手艺再好也只能多赚几条烟钱，人们从来不当他们是回事。

老二是乡下有名的泥水匠，包工头，牛皮的不得了，乡下方圆数十里的老乡建房都找他，他不画图，连文字描述都省掉，一杯茶的工夫就可以和别人敲定一栋小楼房的建设，第二天就可以动工，白天黑夜的干上数月能赚几千元。

老兄，我和你一样都是程序员，IT 业的水泥匠，在乡下人面前可以牛皮，其实连建筑业的施工员都不如啊，施工员还能看懂图纸。你想当一辈子泥水匠吗？虽然工作不分贵贱，但对人类的贡献就是不同，收入就是有高低，我们一起努力吧，泥水匠兄弟，世界上的泥水匠总比工程师多，当你知道了外面的世界，在乡下没钱的老乡面前牛皮是穷牛皮啊！最悲哀的是井底蛙呱呱叫了。

中国 IT 泥水匠：piaoyun 与你共勉。

看清华大学出版社的《软件能力成熟度模型》，终于明白中国软件业什么也不缺，就缺“笨蛋”

piaoyun：

请看第二页：吸取对我国有用的东西，提出了一个符合我国国情的和软件开发水平的软件能力成熟度模型 CSCMM。于是 CMM 变成了 CSCMM。中国留日学生总说日本人笨不灵活，于是日本笨的好富，中国灵的好穷。说起印度人就更不屑一顾了。日本人学东西先照搬，然后为善改，中国人还没有弄懂，就聪明的把人家的东西先吸取精华、去其糟粕。不要怪我刻薄，CMM 弄不出一个 3 级 4 级的能说懂吗？按 5 分算，3 才及格啊！！！有资格来个 CSCMM 吗？有这样的清华学术泰斗，就有这样的中国软件业。清华学子们不要抨击我，我口才肯定不行，请不要说客观条件，这个中国人最会，弄不出个 3、4 来，说的再多也等于放屁，人家有钱就是不给你赚，宁愿找印度的傻瓜。

Ygpfr：

对不起，我觉得你好象没有看完这本书吧。

Shibaoxu：

老兄大可不必如此激动，如果书中的内容对自己没有帮助弃之即可。我个人这本书尚可。

Piaoyun：

呵呵~~谢谢！谁说书差了，反感 CSCMM！幸好这本书除了 CSCMM 这几个字母，还没有多少自己的“特色”。

Youduofu：

病毒特色。其中好象提到了，软件发布之前一定要经过杀毒软件处理。

mymailwh@btamail.net.cn：

其实说来，这本书就是将 cmm1.1 翻译为中文，为了避免版权所以才加上了一个 0 级，画蛇添足！！！！

哎，要不是公司的 cmm 我才不会买这么破的书，浪费 RMB

kjxou:

买这本书的目的，就是中文看起来快一点，爽一点，也怕看了洋文有什么误解的地方。有本中文的好对照一下。至于什么 CSCMM 的，个人也认为是其中糟粕。

Arfayr：

NOD,NOD,的确阿。人家的东西你还没用，就说不合适？搞创新啊，也不能这么没水平阿。中国有几家过三级的？凭什么来修改？

我估计是，改称国家标准，再来一个什么什么部门，于是乎，又有一大堆人升了，发了，中国也出了 N 个什么 CSCMM 五星级软件公司，真的 faint 了。还是踏踏实实慢慢来吧，搞点实在的东西。

Hediant：

老兄何必为一些“毛孩子”大动干戈啊：）

我在上海的时候有“幸”看到了这本书，看了两页，急忙丢掉，老兄何必打动干戈：）

吧现在的水平和“老子天下第一”的圣贤思想比做“毛孩子”也许再恰当不过了：）

davidqql：

所谓聪明，乃小聪明尔，而印度的软件发达却是泱泱之大中华帝国所不能匹敌，当然，其软件和高科技之雄尚不足以使之摆脱贫穷。

反观中国之教授等诸人，不把一个简单的问题搅得很复杂誓不罢休——否则何以体现其水平之高呢？ 曾有朋友说，日本人很笨，拿到一个系统、设备、总是参照手册一步、一步往下做，不敢有丝毫创新，不象吾国吾民，善于从别人的框框中跳出来，自成体系——呜呼！中华何处不英才？创新梦里却徘徊。

Caozq：

这还算客气的，好歹还取其精华，懂得借鉴。

君不见，中国程序员的劣根性：一旦见着别人的什么新鲜的，先一棍子打倒：你这不行，有什么呀？看我的。等这位仁兄的产品出来了，更是一笑大方。比如，XXCASE 工具号称 UML“出发点唐突，不符合实际”，本工具能够如何如何克服 UML 的缺点，包治百病，等到 Down 下一看，不过如此。

Fqnewman：

其实中国人也不必这么自卑嘛，你有没有仔细对比过中印软件产业的收入来源呢？上次不小心看了一下，印度连续几年国内软件产业都是持平的，而出口的量很大，而中国没年软件产值上升幅度都很大（产值大于印度，好象还翻了个倍），但是出口很小，而印度软件先是来料加工，然后才有自有品牌，前者触动后者。印度对于美国等国家的软件厂商来说具有人力资源便宜，语言相同。而在初期中国的人力资源肯定是高于印度的(现在不知道了)，而且语言不通。在中国英语好的大部分都进外企拿高薪,人力资源当然不便宜。在说印度之所以有多家企业通过 CMM5 级，一个主要的因素是 CMM 是外国企业进入美国的门票，告诉我，你们中有多少是做出口的？如果你们不做出口的话，单是把 CMM 做为管理方案代价太高昂，鼎新过一个 CMM2 都要一百万，有这钱你给我好了，包你管得井井有条，加强管理是一定要做的，CMM 不是一定要过的。CMM 源于美国，但是美国最有影响的软件企业 Microsoft 就没有采用他嘛，但并不是说 Microsoft 就没有管理了，其实大家只要有管理的意识，找出一个最适合自己公司的管理方法就行了，把跟风的力气省下来，多做点实事不是很好吗？

Ygpfr：

不是吧。微软没有去过，并不代表他不参看。你可以看那本书《微软项目求生法则》的序言。