什么是用例和用例描述

我发现，在 OO 和 UML 几乎一统天下的今天，仍有很多系统分析员对 OO 和 UML 一知半解，甚至包括很多已经使用了很久 UML 的系统分析员。

于是打算写一个系列文章，将多年来的工作经验做一个总结。对初学者起个启蒙作用，也希望抛砖引喻，与各路大虾共同探讨，共同提高。

这个系列文章将以我对 OO 和系统分析的理解为主，从 UML 基础开始，阐述面向对象的需求分析方法，过程，并以 RUP 为例，阐述如何将 OO 过程与软件过程有机结合在一起，做一个真正 OO 应用。

好了，今天是第一篇。想得很远，不知能否坚持下去，呵呵:lol:

用例是什么？其原始英文是 usecase，直译过来就成了用例。这也是一个比较贴切的叫法了，从字面的直接理解就是使用的例子。另一种比较流行的定义是用例就是与使用者(actor)交互的，并且给使用者提供可观测的有意义的结果的一系列活动的集合。

这个定义还是比较费解的，笔者在众多应聘者中发现很多使用用例来做需求的系统分析员，有的已经使用了两年以上，但仍不能把握用例的本质，虽然他们号称精通 UML。

最具普遍意义的理解错误是认为用例就是功能的划分和描述，认为一个用例就是一个功能点。在这种理解下，用例变成了仅仅是较早前需求中功能框图的翻版，很多人用用例来划分子系统，功能模块和功能点。如果这样，用例根本没有存在的必要。有意思的是，造成这种理解错误的相当一部分原因却是因为对 OO 思想的理解不够深入，本质上说，把用例当成功能点的系统分析员脑子里还是面向过程的那一套思想，虽然他们在使用 OO 的工具，OO 的语言，号称在做面向对象的开发，但过程的影子还没有从他们脑子里彻底抹去。

如果用例不是功能的话，它是什么呢？从定义上说，能给使用者提供一个执行结果的活动，不就是功能吗？我的回答是：错！功能是计算机术语，它是用来描述计算机的，而非定义需求的术语。功能实际描述的是输入-->计算-->输出。这让你想到了什么？DFD 图？这可是典型的面向过程分析模式。因此我说把用例当做功能点的分析员实际在做面向过程的分析。

而用例则不是计算机术语，UML 除了在计算机行业中应用，它也经常被应用在其它行业中。用例是一种需求方法学，虽然软件危机和 OO 的发展促成了它的诞生并被完美的融合进了 OO 体系，形成了 UML，但它实际上并不是软件行业的专用品。如果非要从功能的角度解释，那么用例可以解释为一系列完成一个特定目标的“功能”的组合，针对不同的应用场景，这些“功能”体现不同的组合方式。实际上，把用例解释为某个参与者(actor)要做的一件事可能更为合适。这样的一件事有以下几个特征：

一、这件事是相对独立的。这意味着它不需要与其它用例交互而独自完成参与者的目的。也就是说这件事从“功能”上说是完备的。读者可能会想到，用例之间不是也有关联关系吗？比如扩展，比如实现，比如继承，它看上去并不是独立的嘛。关于这个问题，笔者会在后续的文章里详细说明。这里稍微解释一下，用例之间的关系是分析过程的产物，而且这种关系一般的产生在概念层用例阶段和系统层用例阶段。对于业务用例，这个特征是很明显的。

二、这件事的执行结果对参与者来说是可观测的和有意义的。例如，系统会监控参与者在系统里的操作，并在参与者删除数据之前备份。虽然它是系统的一个必需组成部分，但它在需求阶段却不应该作为用例出现。因为这是一个后台进程，对参与者来说是不可观测的，它应该在系统用例分析阶段定义。又比如说，登录系统是一个有效的用例，但输入密码却不是。这是因为登录系统对参与者是有意义的，这样他可以获得身份认证和授权，但输入密码却是没有意义的，输入完了呢？有什么结果吗？

三、这件事必须由一个参与者发起。不存在没有参与者的用例，用例不应该自动启动，也不应该主动启动另一个用例。用例总是由一个参与者发起，并且满足特征二。例如从 ATM 取钱是一个有效的用例，ATM 吐钞却不是。因为 ATM 是不会无缘无故吐钞的，否则，我从此天天守在 ATM 旁，生活无忧矣。

四、这件事必然是以动宾短语形式出现的。即，这件事必须有一个动作和动作的受体。例如，喝水是一个有效的用例，而“喝”和“水”却不是。虽然生活常识告诉我们，在没有水的情况下人是不会做出喝这个动作的，水也必然是喝进去的，而不是滑进去的，但是笔者所见的很多用例中类似“计算”，“统计”，“报表”，“输出”，“录入”之类的并不在少数。

除去以上的特征，笔者觉得用例的含义还要更深些。首先，用例的背后是一种需求方法论。其核心是以参与者为中心（区别于以计算机系统为中心），从参与者的角度来描述他要做的日常工作（区别于以业务流程描述的方式），并分析这些日常工作之间是如何交互的（区别于数据流的描述方式）。换句话说，用例分析的首要目标不是要弄清楚某项业务是如何一步一步完成的，而是要弄清楚有多少参与者？每个参与者都做什么？业务流程分析则是后续的工作了。其次，用例简直就是为 OO 而生的，其思想完美的符合 OO。用例分析方法试图找到问题领域内所有相对独立的参与者和事件，并把业务流程当成是这些参与者和事件之间的交互结果（在 UML 用活动图或序列图来描述）。因此，用例方法被吸纳到 OO 之后，UML 得以以完备的形式出现，用例成为了真正的 OO 核心。在 RUP 里，这种核心作用被发挥到极致，产生了用例驱动（usecase driven）的软件过程方法，在 RUP 里，软件生产的所有过程和产物都是围绕着用例形成的。

可以说，用例分析是 OO 的第一步。如果用例分析本身出了问题，对业务架构，软件架构的影响是很大的，将大大削弱 OO 的优势--复用、扩展。笔者认为软件复用可以分为三个层次，最低层次的复用是代码级复用，这是由 OO 语言特性提供支持的，例如继承，聚合，多态；较高层次的复用是组件级复用，这是由设计模式提供支持的，例如 Factory 模式,Builder 模式；最高层次的复用则是服务级复用，这在很大程度上是由应用服务器和通讯协议来提供支持的，例如最近炒得火热的 SOA（面向服务的应用）架构。用例分析的好坏也许对代码级和组件级的复用影响不太大，但对服务级的复用影响却是巨大的。笔者认为服务级复用是 OO 的最高境界，而结构良好的用例分析则是达到这一境界的基础。

闲话：

今天你 OO 了吗？

如果你的分析习惯是在调研需求时最先弄清楚有多少业务流程，先画出业务流程图，然后顺藤摸瓜，找出业务流程中每一步骤的参与部门或岗位，弄清楚在这一步参与者所做的事情和填写表单的结果，并关心用户是如何把这份表单传给到下一个环节的。那么很不幸，你还在做面向过程的事情。

如果你的分析习惯是在调研需求时最先弄清楚有多少部门，多少岗位，然后找到每一个岗位的业务代表，问他们类似的问题：你平时都做什么？这件事是谁交办的？做完了你需要通知或传达给谁吗？做这件事情你都需要填写些什么表格吗？....那么恭喜你，你已经 OO 啦！

用例组成

当记录基于组件的系统的行为需求时，用例是最常用的技术之一。开发人员常问的一个问题是，“用例文档应该包括哪些信息？”尽管我在此提到的一些部分是可选的，但在我看来，将这些部分包括在用例文档中不失为一个好主意。当编写基本用例的文档时（另请参阅前一篇技巧 Modelling essential use cases），我倾向于略去可选部分（因为基本用例关注的是是什么，而不是 为什么，因此不必像系统用例那样复杂）。当编写系统用例时，我通常将所有部分都包括在内。回顾一下，基本用例和系统用例之间的主要区别是，系统用例包括了高级实现决策，而基本用例是要以与技术和实现无关的方式捕捉用户的意图。

参与者 (actor) 和被包含的用例这两个部分实际上只看用例图即可确定。但是，按我的经验，各个用例最好相互独立 — 换句话说，用例应该包含理解它们所需的全部关键信息以及它们所在的上下文。这使您的主题问题专家 (SME) 能够分别充实各个用例。（他们可能上午以小组为单位协同工作，下午则各自独立地以最快的速度充实所分配的用例，从而提高了整个小组的生产效率。）

用例的各个组成部分

名称。名称无疑应该表明用户的意图或用例的用途，如“研究班招生”。
标识符 [可选]。唯一标识符，如 "UC1701"，在项目的其他元素（如类模型）中可用它来引用这个用例。
说明。概述用例的几句话。
参与者 [可选]。与此用例相关的参与者列表。尽管这则信息包含在用例本身中，但在没有用例图时，它有助于增加对该用例的理解。
状态 [可选]。指示用例的状态，通常为以下几种之一：进行中、等待审查、通过审查或未通过审查。
频率。参与者访问此用例的频率。这是一个自由式问题，如用户每次录访问一次或每月一次。
前置条件。一个条件列表，如果其中包含条件，则这些条件必须在访问用例之前得到满足。
后置条件。一个条件列表，如果其中包含条件，则这些条件将在用例成功完成以后得到满足。
被扩展的用例 [可选]。此用例所扩展的用例（如果存在）。扩展关联是一种广义关系，其中扩展用例接续基用例的行为。这是通过扩展用例向基用例的操作序列中插入附加的操作序列来实现的。这总是使用带有 <<extend>> 的用例关联来建模的。
被包含的用例 [可选]。此用例所包含用例的列表。包含关联是一种广义关系，它表明对处于另一个用例之中的用例所描述的行为的包含关系。这总是使用带有 <<include>> 的用例关联来建模的。也称为使用或具有 (has-a) 关系。
假设 [可选]。对编写此用例时所创建的域的任何重要假设。您应该在一定的时候检验这些

假设，或者将它们变为决策的一部分（请参阅下文），或者将它们添加到操作的基本流程或可选流程中。
基本操作流程。参与者在用例中所遵循的主逻辑路径。因为它描述了当各项工作都正常进行时用例的工作方式，所以通常称其为 适当路径 (happy path) 或主路径 (main path) 。
可选操作流程。用例中很少使用的逻辑路径，那些在变更工作方式、出现异常或发生错误的情况下所遵循的路径。
修改历史记录 [可选]。关于用例的修改时间、修改原因和修改人的详细信息。
问题 [可选]。如果存在，则为与此用例的开发相关的问题或操作项目的列表。
决策。关键决策的列表，这些决策通常由您的 SME 作出，并属于用例的内容。将这些决策记录下来对于维护团体记忆库 (group memory) 是相当重要的。