用例有粒度吗

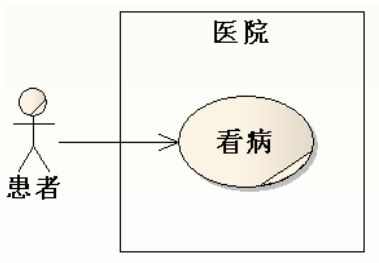
潘加宇

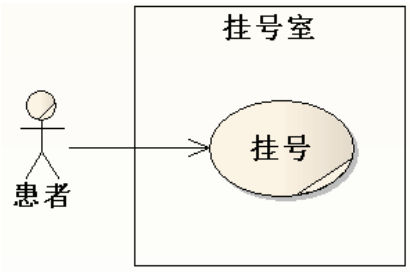
用例真难！这么简单的技术为什么会让人觉得很难？用例其实很简单，就是实事求是说人话，表达出系统的契约和价值。而做到这一点并不容易，现实生活中我们就很难做到实事求是说人话，习惯于把我们真正的目的层层包裹起来，实践用例的过程类似于揭开脓包，掀开地毯，当然会让人感到不适。

什么东西的用例？

很多时候我们讨论得很热烈，用例来，用例去。就是忘了最基本的东西，到底在讨论什么东西的用例？或者说，研究对象是什么？因为用例是某个东西对外承诺的价值，没有这个前提，讨论将失去意义。

比如说，“挂号”算不算用例？就要看你的研究对象是什么。如果研究对象是“医院”，患者到医院来，挂到了号，医院的服务到此为止，患者对医院的期望满足了吗，没有。（你说有？那你可能不是患者，是黄牛党）。或者说，医院这个东西本身能承诺向患者提供的价值不只是挂号。如果研究对象是挂号室，就不一样了。患者对挂号室的期望就是能挂号，他不会因为挂号室没帮他看病就破口大骂。如果研究对象是医院信息系统，又不一样了，患者无所谓挂号室人员是怎样帮他挂号的，医院信息系统的价值是让挂号室人员通过它来（为患者）办理挂号。所以以下几个是正确的（小人和椭圆上有斜杠的，表示研究对象是一个业务单元，相应的执行者和用例为业务执行者和业务用例）：





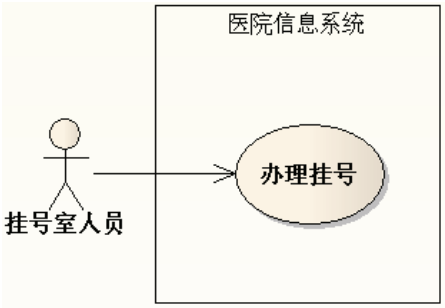


图 1 正确的用例图

而以下是错误的：

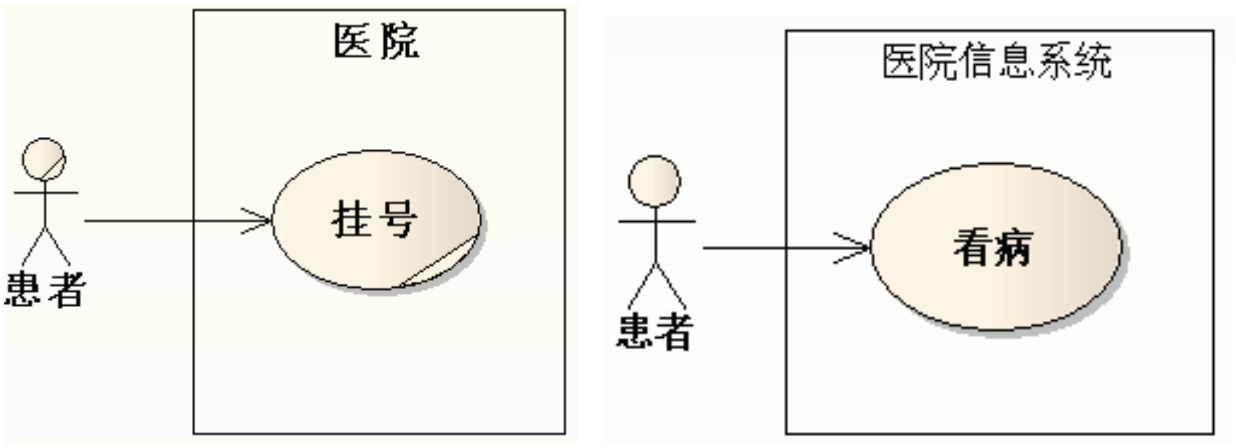


图 2 错误的用例图

您可能已经注意到了，每幅用例图都有一个边界框来标明当前的研究对象是什么，如果有的工具没有提供这个框，最好也要用一个 Note 注明研究对象。例如 Rose 里：

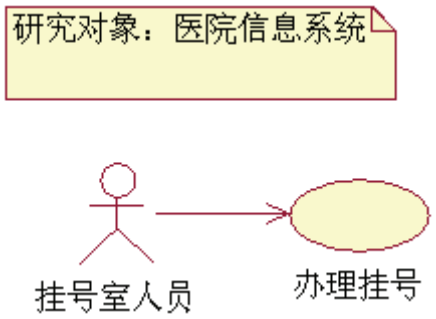


图 3 用 Note 注明研究对象

如果之前有业务建模，画出业务序列图来，分清楚你干什么我干什么，这种困惑就迎刃而解了。从图 4 我们可以很清楚地知道，医院信息系统真正的用例是“挂号室人员→办理挂号”

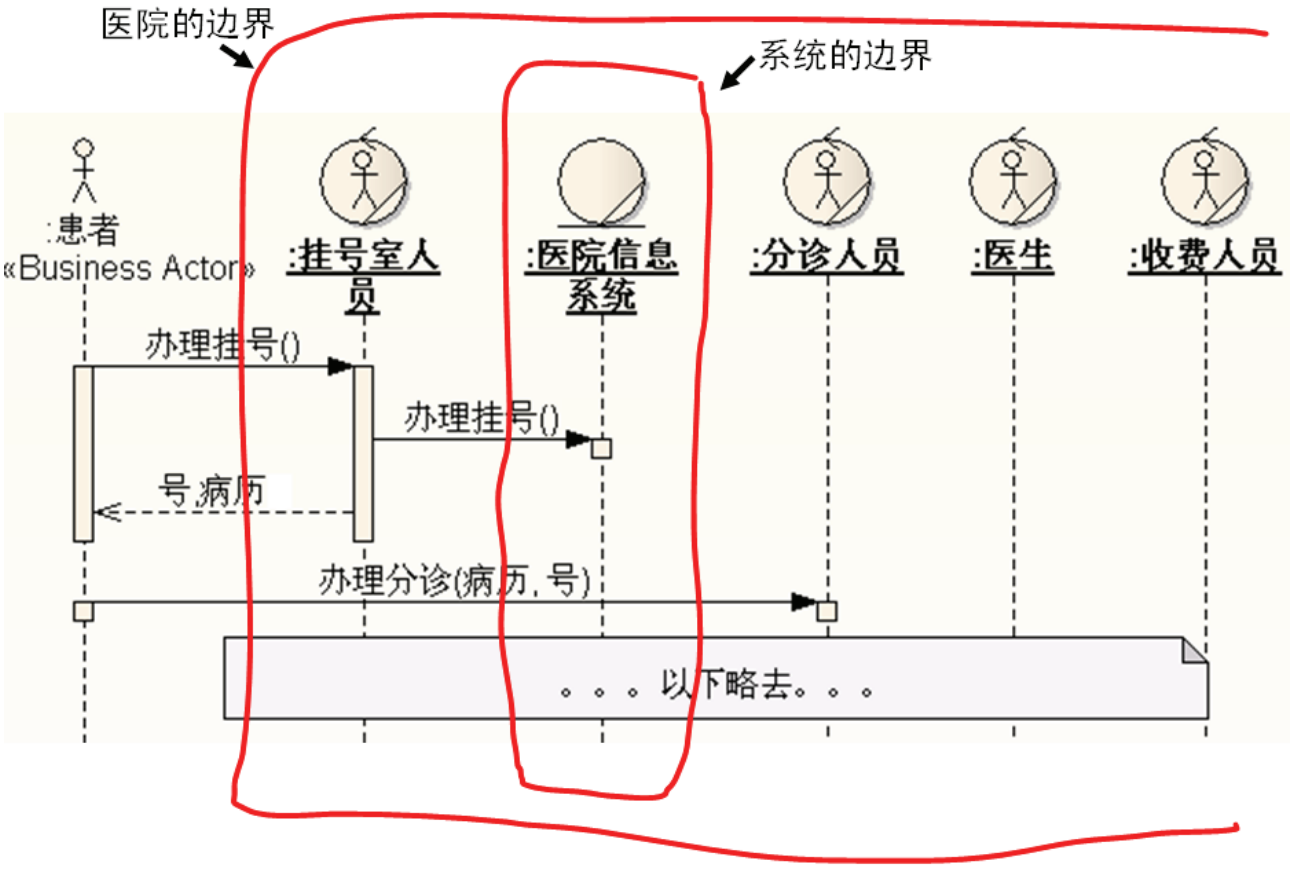


图 4 医院的“患者－－>看病”用例的业务序列图

“粒度”问题

理解了上面这个，“粒度”的困惑就迎刃而解了。注意，笔者加了双引号，也就是说，所谓“粒度”，其实并不存在。用例不是面团，任由开发人员来切割，开发人员只能根据涉众心中对系统的期望，最后确定系统能提供什么，不能提供什么，并把它实事求是地写下来。

“登录”是用例吗？如果研究对象是一个在线购物系统，严格来说它不是。如果研究对象是一个身份识别系统，“登录”作为用例就是合适的。如果研究对象是一个输入密码的设备，那么连“输入密码”都可以作为用例。

只要在形式上能写出符合需求标准的路径、步骤，都可以作为用例。除此之外，“粒度”是多大，开发人员已经没有资格说话。什么叫“符合需求标准”呢？就是所写的所有东西要能够被涉众理解和验证。

如果说大家在热烈地讨论粒度问题，纠缠不清，那多半是犯了错误。最常犯的错误是：把步骤当作用例。如下图中取款机的用例：

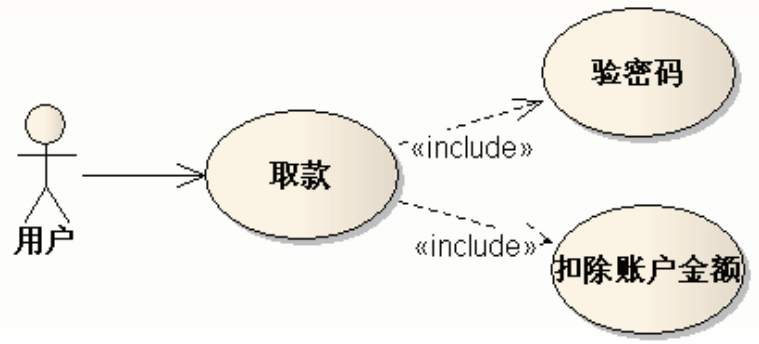


图 5 最常犯错误：把步骤当作用例

右边两个其实只是步骤，不是用例。Include 关系也不是这样的，Include 的目的是为了复用有价值的步骤集合，形状往往是很多个老大 Include 一个小弟。

有的书里面会给出“粒度原则”，例如：一个用例的基本路径最好控制在×步到×步之间、一个系统的用例最好控制在××个之内……对于这些说法要实事求是地对待，不能因为这是某大师说的，就一定是对的。毕竟现在经常被提到的那些经典用例书籍，从出版到现在已经有一些年头了。象 Ivar Jacobson 的“Object-Oriented Software Engineering”[1]出版于 1992 年，Alistair Cockburn 的“Writing Effective Use Cases”[2]出版于 2001 年。不否认这些书中思想的光芒，但在实践的大浪淘沙之下，有些细节值得商议。用例的专著尚且如此，更不用说了起名为《UML**》的那些书籍了。“粒度”、“层次”这些概念迎合了开发人员的“设计癌”，对开发人员的误导相当严重。碰到开发人员玩弄“粒度原则”、“分层技巧”时，笔者有时甚至会感到愤怒，“把屁股坐过那边去，揣摩涉众的心理，实事求是写下来！别管是大还是小！”对于“用例是否用对了”，笔者有一个朴素的判断标准：是否加强了和涉众的联系。如果不是，那就用错了——别管书上怎么说。

其实涉众心里的“最佳答案”还是有倾向性的。看下面的例子，应该选左边的用例图还是右边？

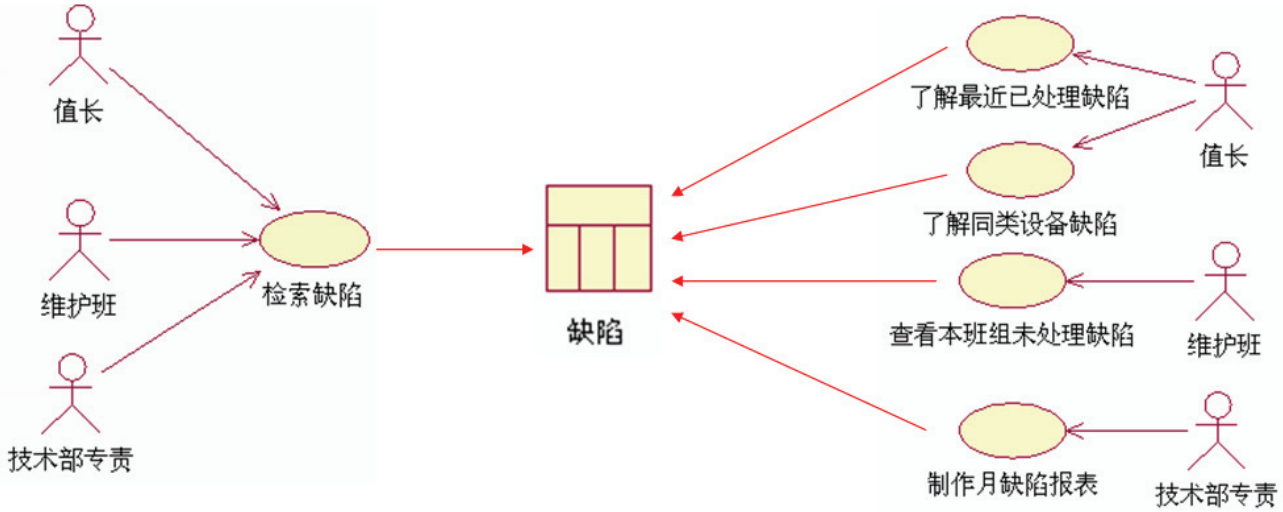


图 6 用例能多勿少

开发人员经常会想，咳，不都是针对“缺陷”表来“Select ××× from 缺陷 where ×××”嘛，合并成一个用例可以少写好多文档，选左边。但实际上用例是客户愿意“购买”的、对系统的一种“用法”，只要客户愿意“购买”，当然是越多越好。同样的制作材料，变出更多可卖的功能，何乐而不为？

下面这个也很常见。一份申请表要经过多次审批。

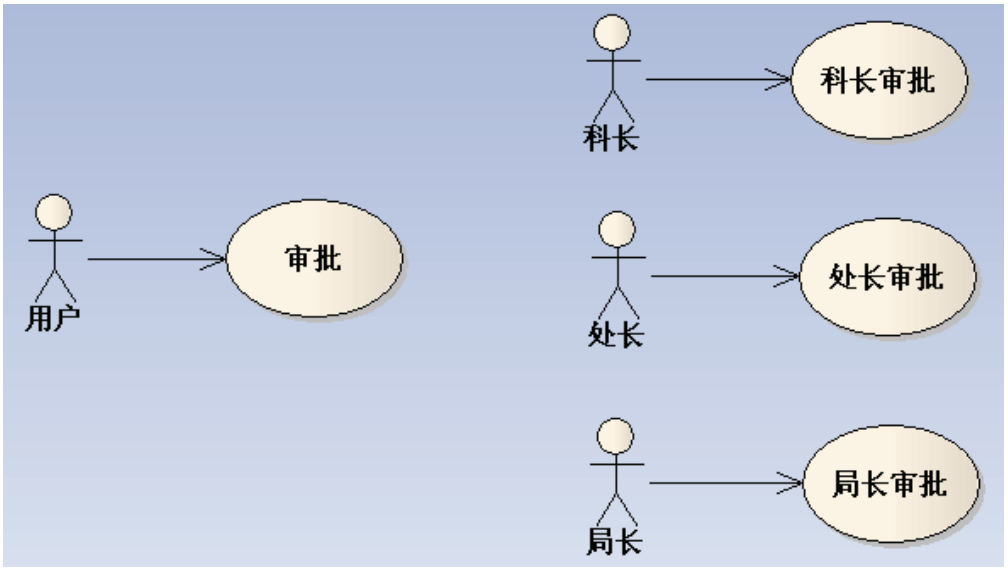


图 7 从涉众的角度看用例

开发人员也常会“通过现象看本质”，可能就是“申请表”对象的某个状态值发生变化而已（其实，往往并非如此简单，因为涉众利益不同），左边的用例图更能体现“本质”。但我们从涉众的角度想一想，局长心里乐意和科长共享一个功能吗？

“层次”问题

和“粒度”相近的是“层次”问题，同样，笔者也加了引号。出现“层次”问题的原因可能是不知不觉偷换了研究对象，或者是把契约和非契约的东西混在一起谈了。象医院系统的用例，有人会画成这样：

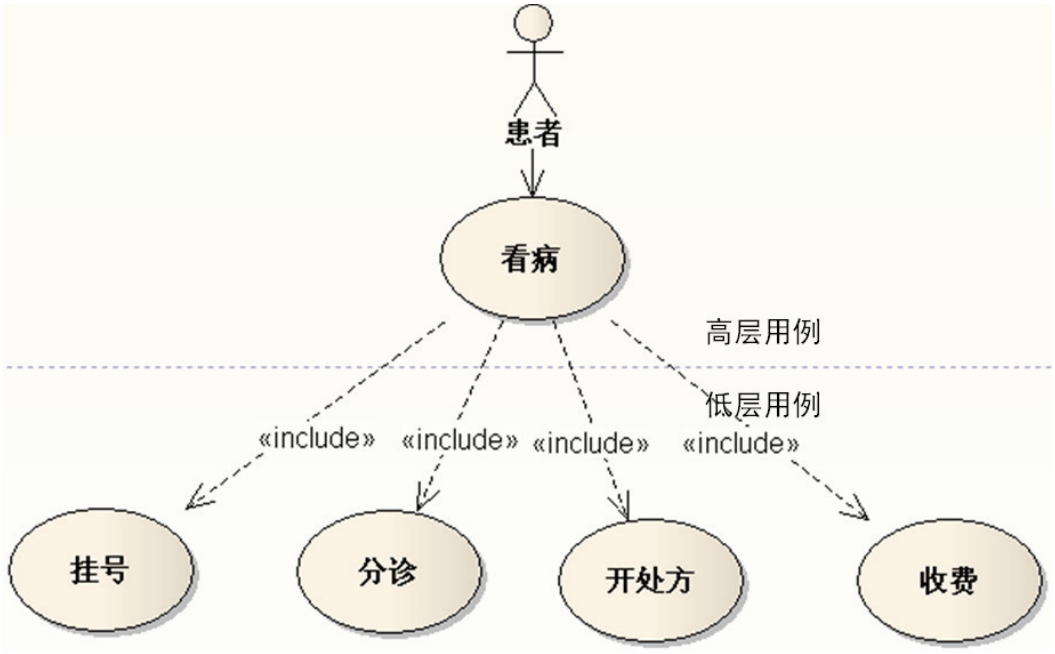


图 8 错误的“高层”用例，偷换了研究对象

这个“高层”实际上是头脑里不知不觉偷换了研究对象，“医院系统”的契约换成“医院”的契约了。还有下面这个：

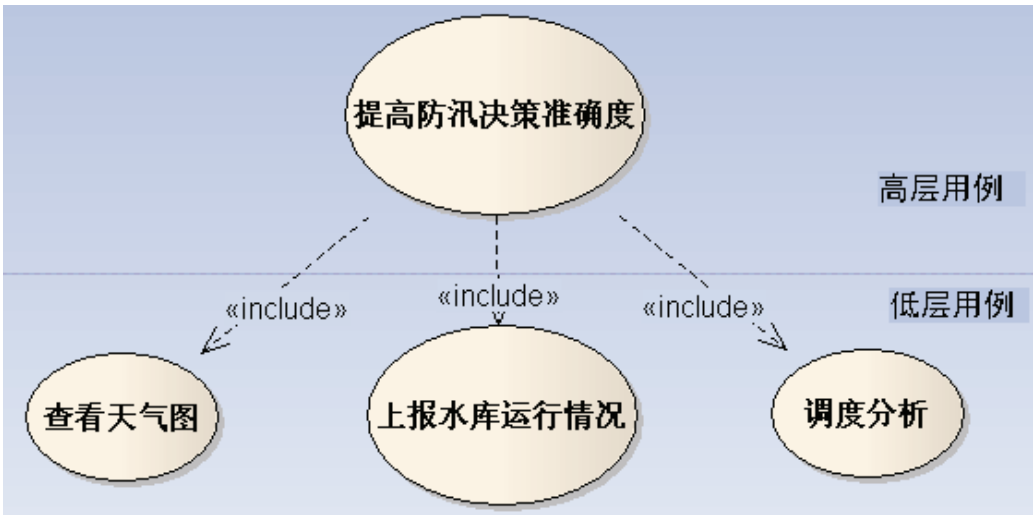


图 9 错误的“高层”用例，把愿望当成契约

“提高防汛决策准确度”只是某类涉众（领导）的一种愿望，就像储户希望取款机“操作方便”一样，不是系统的一个契约。系统没有提供一个按钮，领导一按，防汛决策准确度就提高了。

改进之路

以下几个实践可以帮助我们少犯以上提到的错误：

不要总想着“复用”或“抽象”需求。从机器的角度看，所有软件的“需求”都是 0 和 1，从数据库的角度看，所有的“需求”都是新增修改删除查询。但只有从个人（涉众）的角度看，得到的东西才是真正的需求，它们是多姿多彩、千变万化的。同样是以面为主原料，西北的人民居然能提供如此多的“用例”：拉面、刀削面、擀面、擦面、哨子面、扯面、油泼面…还有 BIANGBIANG 面。同样是手机，市场居然有那么多的品类和品牌可以选择。讲究“复用”不是需求要考虑的事情，而是设计要考虑的。用高焕堂老师的话说，需求是收益面，设计是成本面[3]，需求关心的是怎样“卖”得好，设计关心的是怎样“低成本”大批量制造。

面向对象的业务建模。通过拍脑袋来获得用例，它的“粒度”难免会模糊。如果有了面向对象的业务建模，就会清晰很多。从更大的范围来看，我们要开发的系统只不过是业务组织里面的一个业务对象，系统的用例就是这个对象对外提供的服务，即它的操作。如果我们能够严肃地用业务序列图来描述业务流程，理出待开发系统可以改进的地方，把相应的责任转移到待开发系统上，得到改进后的业务序列图。在这个图上，待开发系统对外提供的服务就是它的系统用例。用例技能和对象技能在思想上是共通的，笔者发现，喜欢犯“把步骤当作用例”错误的开发人员，在设计类的时候，往往也会喜欢让类暴露一个类似“修改状态”的操作。

书写用例文档。如果拿不定主意，可以把这个用例的文档写出来看看。用例的步骤应该是回合制的，一个回合内包括以下几类步骤：1. 执行者请求；2. 系统验证 （可选）；3. 系统改变 （可选）；4. 系统回应。如果写出来的文档第一句话就是“系统×××××”，也许犯了“把步骤当作用例”的错误了（“粒度”太小）；如果写出来的文档类似于“执行者做某事，执行者又做某事，执行者….”，也许是把几个用例并成一个了（“粒度”太大）。

参考资料

[1] Ivar Jacobson, Object Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, 1992/7

[2] Alistair Cockburn 著，王雷、张莉 译，《编写有效用例》，机械工业出版社，2002。

[3] 高焕堂 著，《Use Case 入门与实例》，清华大学出版社，2008。