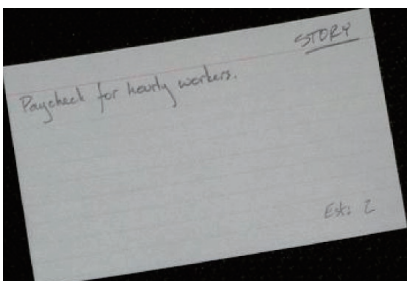
故事卡以外的故事：敏捷需求协作

用户故事是敏捷项目中常用的需求获取技术，然而隐藏在其背后的，是对于产品经理、开发团队、客户坐在一起、紧密协作的要求。

■ 文 / James Shore 译 / 李剑

下面是张故事卡。用它来做计划很赞，不过跟人沟通需求可就差劲得很了。如果你玩过很长时间的极限编程，或许就听人说过，“故事卡可以当作记号，以后用来做沟通”。但这句话到底是什么意思？故事卡是怎么产生的？有了故事卡以后，你又该怎么使用？



一张故事卡

今天，我会讲一下在创建故事卡前后会发生些什么。我唯一不会提到的东西就是故事卡本身。

故事卡以外的故事

我叫Jim Shore，从99年起就开始训练团队实施敏捷方法，当然，那时候我们还把它叫做“轻量级方法”。我一开始是带领团队使用特征驱动开发。第二年试了一下XP，结果很成功，然后就一直帮着别人实施XP。我在很多公司工作过，这让我了解到XP中哪些实践管用，哪些不能。

整理需求的方式很多，整理敏捷需求的方式也不少。下面要谈的内容，是从我过去几年实施XP的经验中提炼出来的。它对当前存在的需求工程技术是一个补充，你可以把它跟其他任何一种你正在使用的方法结合。

为何协作？

这次演讲的题目是“敏捷需求协作”。选这个题目就费了我很大功夫。我本来可以用“敏捷需求收集”，或是“敏捷需求工程”，但是最后还是用了“协作”，因为用户参与是软件项目成功的核心要素。无论项目是不是敏捷。敏捷开发推崇协作，因为它可以提高成功的机会。这里我需要强调一下，我下面要讲的是怎样增强协作，而不是把需求弄出来然后就扔到墙上。

故事从何而来？

故事一开始从哪儿来呢？呃……是人写出来的。某些人一定要对产品的模样有个整体认识。在XP项目里面，这个人通常被人们叫做“现场客户”，有的也叫“关键客户”或“产品经理”。我这里会把它叫做“产品经理”，因为这个词更贴近于常见的行业用语。

软件项目的成败在很大程度上取决于项目经理

产品经理需要对产品有清晰的认识，不过这并不意味着他要自己对产品负责。产品经理会跟交互设计师、业务分析师、真实客户、市场人员、用户等等一起工作。到最后，要有人把这些五花八门的观点收集起来，形成明确的产品描述。这就是产品经理的职责。

产品经理的工作可不轻松。他必须持有强有力的观点，能够把所有意见紧紧收束到一起，得出功能强劲的产品；他要对市场有透彻的理解，了解当前所需；他又必须会左右逢源，调节所有相关人等的要求；最后，他还得有足够的权威，他定下来的优先级不会被别人改变。

这种人对公司简直是无价之宝，要借到他们的时间难得很。在我亲见的XP团队中，最大的难题就是怎么弄到一个优秀的产品经理。对这种情况我要说的就是：没错！好的产品经理很值钱。但我们也要记住，要想项目成功，他们的参与就至关重要。开发软件的成本比产品经理贵得多。要是你的产品不值得付出一个优秀产品经理的时间，也许你就该重头考虑一下，这个软件值不值得开发。

听众提问：如果没有人对产品有整体理解怎么办？

我从没碰到过。有时候人们不知道他们想要什么。

他们以为自己知道，但真正看到产品运行的时候，他们才意识到原来他们想要的是另外的东西。我也看到过有人很难描述出自己想要的东西，对于这种情况，我们就可以一个迭代接一个迭代的频繁部署，这种敏捷方法可以帮助人们理解他们的真正需要，因为他们可以不断看到实实在在运行的软件。

我曾经在一个团队中待过，我们跟客户一起工作，每两个星期就给他看看产品。我们不太确定交付的东西是不是他想要的。于是我们每次都会说："我们要部署这些！"他就回答我们："很好，很好。"然后，离交付还有四星期的时候，我们又重复了同样的话，他突然说："等一下！这是我们要交付的东西？不，不，全错了。"

但是我们只用了两个星期的迭代就改成了他想要的模样。大多数变化只是改一下用户界面而已。

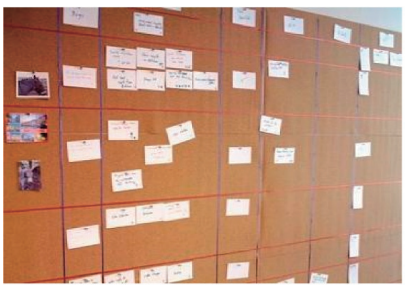
听众提问：我觉得更常见的情况就是，有很多人都对产品有自己的想法。你是怎么平衡的？

这就是产品经理的职责了。他需要平衡各方的利益冲突。这人肯定得会左右逢源才行。我从前试过不少方法，比如给每一组都分配一些迭代预算，或者让他们自己判断自己故事的价值，但效果都不太好。我们在大量琐事上花了太多时间，结果却一事无成。实际上，只要有个产品经理来做决定就好了。

最后负责时刻

产品经理在团队帮助下，与业务分析师、客户、市场人员，一起编写故事。但是让我们注意一下"故事墙"。

等你有了300张故事卡，要一一跟踪状态以后，故事墙就成了地狱。故事卡不是用来捕捉产品需求的全部细节，它只应该由一两句话组成，充其量是一张图。要是有了300张卡，你就再没法理解它们到底是啥意思。假设有张卡上面写的是"弄早点"，另



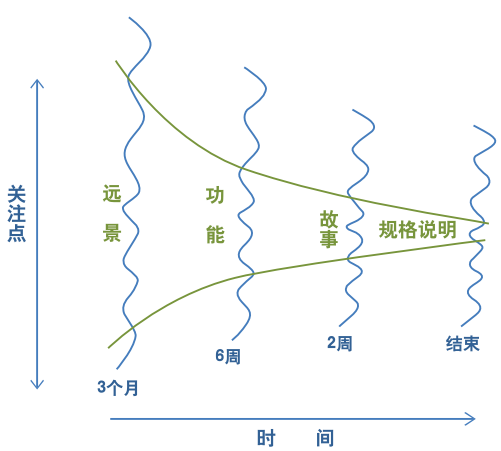
故事墙

一张卡上写着"做早饭"，这两张卡说的是一个故事吗？还是完全不同的两个部分对应的不同需求？你没法判断。在树木重重包围下，你也看不到森林的全景。

一旦你听到这种问题，"我们能不能把这些东西放到数据库里面去？"那就证明你已深陷故事卡的无边地狱。我服务过的每个客户都想把故事卡放到数据库里面去，这完全偏离了故事卡的本意。当你开始担心故事卡会丢失，也就是用的过头的时候。

这让我们想到了"最后负责时刻"理论，也是敏捷的核心思想。这个名字来自于精益软件开发。它的含义是在尽可能的情况下，直到还可以为事情负责的最后一刻再做决定。这里说的是能够负责的最后时刻，而不是真正的最后一刻，那不可能做到负责任。

这样一来，将决策付诸实施的成本就更加低廉，成果也更为显著。明天所花出去的钱就比今天便宜，而且在这一天内，我们还会掌握到更多当



细节层次和被实现时间的关系

前环境的信息，决策也会更完善。而且世事变幻，沧海桑田，今天的决定，明天未必还能有效。花的时间精力就白白浪费掉了。所以只要我们还能为事情负责，我们就会一直等待。

关注愈来愈多的细节

使用"最后负责时刻"的思想，我们把创建详细需求规范的时刻推迟了。实际上，我们可以在实现故事的时候再去编写它的规范说明（我会稍稍加以详细说明的）。我们不需要提前知道每一处细节。细致入微代价昂贵，一旦计划有变，努力便付诸流水。

在上面的图里面，我们可以看到细节层次和距离被实现的时间远近的关系。在大多数敏捷方法中，我们都喜欢把发布——起码是内部发布——的时间限制在三个月以内。所以如果交付还远在三个月之外，你所需要的就只是一个宏观理解而已。这款软件面向什么样的市场，它要提供何种价值，它的基本功能是什么。也许你需要有一个产品概述文档或是任务说明。

走进三个月以内，你开始需要计划一下哪些特性需要发布。细节进一步完善。有些人把它们叫做"发布级故事"、"高段故事"、"孕育期故事"。不管叫什么，你都得定义出来产品的主要特性。你可以把它们写到故事卡上。

六星期是做详细计划的一个好起点。这是能够预测细节的最远距离。你可以借助故事卡进行详细计划，所以一旦有了一个距离发布不足六星期的特性，就可以开始研究细节创建故事了。如果市场和产品需求比较稳定，这个期限还可以长一点，如果一切杂乱无章，可能就得短些。

当然，一旦开始了故事的实现之旅——这张图里，我假设迭代周期是两周，所以这个时间是距离发布两周之前——你就需要整个故事的所有细节。这就是编

写规范的时机。

这是个理想化的世界。在实践中，我们不可能从宏观理解，到特性，到故事，到规范一路势如破竹，恰恰符合这里描述的时间段。现实总是比理想麻烦一些，我们很难知道什么时候可以去深究细节。但是整体方向还是正确的：你不必预先知道所有细节。在需要的时候——最后负责时刻——再去深究细节，就可以远离苦海。

听众提问：你怎么定义“特性”？

怎么定义都行。这里的重点是随着时间推移，慢慢增加细节。我喜欢Software by Numbers里面的定义，它用的是“最小可市场化功能”：可以给市场带来价值的最小集合。你可以把特性当作产品经理要求团队完成的任何一样东西。

听众提问：你能在超过三个月之前来定义产品远景么？

当然。三个月是我开始关注更多细节，定义特性的时候。如果你的项目周期很长，大约一两年的样子，当然需要对整个项目有宏观把握。也许等你越过三个月这条线以后，整体规划就会更加明确，也有了具体的特性。

缺少用户参与一度是项目失败的首要原因。反过来说，用户参与在项目成功的因素中高居榜首。即使能够做到按时提交，不超出预算，项目也会因为不符合用户需要或偏离用户期待而失败。

协作奇迹

如果产品经理和他的团队可以跟开发人员从始至终直接协作，就会有神奇的结果产生。我把它叫做“协作奇迹”。没错，确实是奇迹，因为每次都是突然就多出来一些时间。

不过达到这条件也并不容易。团队需要良好协作。我们先看一下这种情况，产品经理跟开发人员一起工作，开发人员给出估算，产品经理给的问题让所有编码的人都开始咬牙切齿，“为什么成本这么高？”

人们对这个问题的本能反应是自我防护。“我说，成本高是因为软件开发很困难，你问这干嘛！”

但是也可以演化成另外一种情况。虽然不太容易。不过你可以试着接过“为什么成本这么高？”这个问题，然后换个方式思考。你可以把它当作是有人在诚恳的要求获取信息：“它为什么成本高呢？”你可以讲一讲哪些地方做起来容易，哪些地方做起来困难，这就回答了他的问题。

例如，产品经理要求烤面包机可以在面包烤好以后自动把面包弹出来，开发人员可能会说这成本很高。产品经理问为什么，开发人员回答说，“嗯，把面包弹出来很容易，就是个弹簧而已。但是检测面包什么时候烤好——这就难了。我们需要一个光传感器，还需要自定制一些检测褐色的软件。”

产品经理就有机会继续问：“但是其他现有的烤面包机怎么样呢？它们知不知道面包什么时候烤好呢？”

开发人员回答说：“它们用得是定时器。但是并不能真正检测到面包有没有烤好。就是凑合着用罢了。”

产品经理会说，“那就够了！我们的客户不需要一台超级烤面包机。他们只想要普通的。就跟其他厂家一样用定时器吧。”

然后开发人员说，“好啊。那成本就低了。”

如果你这样做了，如果产品经理和开发人员可以坦诚交流，协作奇迹就会产生，突然就有了多余的时间。这是因为产品经理不知道哪些东西好做，哪些东西难做。他们也许以为自己知道，他们也许会猜，但他们常常都错得很离谱。

同样，开发人员也不知道产品经理的哪些想法比较重要。他们也许以为自己知道，他们也许会猜，但他们也常常都错得很离谱。

有时这两个案例会有交叉——不是时时如此，但频率也比较高，值得我们关注。产品经理要求一些很困难但不是特别重要的功能。他并不知道要完成这功能很困难，所以他就提出了要求。如果开发人员说很困难，而且提出了简便的替换方案，产品经理就会改变方向，从而节省时间。不是每个故事都会发生这种情况，但也是经常出现，我们得统计进去才行。突然就多出了时间。这就是协作奇迹。

听众提问：如果开发人员不能给出可靠的估算怎么办？

我发现开发人员估算范围很棒，但是估算时间不行。他们的估算可能有问题，但还是能保持一致的，所以能够得出某个功能比另外一个难两倍的结论。估算故事比估算特性或者整体发布的准确性更高。

如果开发人员做不出前后一致的估算，这可能是由于更大问题导致的。我估计他们面对着许多设计债。要是牺牲了代码质量来迎合最后期限，就会留下设计债。你这是从你自己的代码中借账，虽然你目前得到了一些自由时间，但是到了后期，所有功能都需要多花一些时间才能实现，因为代码质量已经没有了。这就是给你自己的时间账付利息——利率还很高。解决方案只有一条，把原来受到的损失补回来——花时间修补那些权宜之计。

不过还是有人意识不到自己在犯下设计债。所以债台高筑，代码质量严重恶化。做事速度越来越慢。每件事情都比预期用的时间更长，开发人员再也没能力做出估算，因为他们总

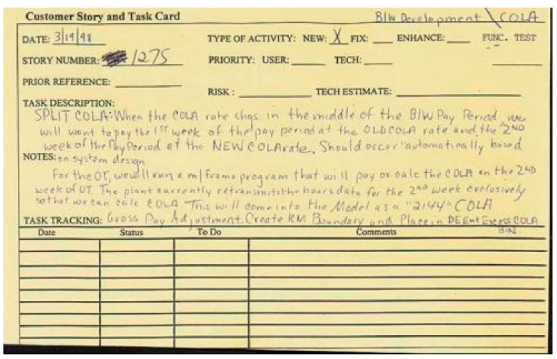
是碰到预料不到的问题。

如果开发人员无法给出可靠的估算，你就可能遇到了设计债。你需要花时间重构，给欠债买单。不过不要一次搞完——一边实现新特性，一边做重构。

听众提问：什么情况下重写比重构效果更好？

我不愿意重写。风险高，花的时间也比预计的长。在很长一段时间内，你都无法给客户提供任何价值，因为你在重新实现已有的功能。

从另一个角度来说，软件也可能糟到无可复加的地步，以至于重写的代价比重构小。但我不知道这界限在哪里。对重构了解的越多，我就越倾向于重构，而非重写。Luke Hohmann说过一句话，这也是你应该记住的第一条规则，迄今为止，重写软件花去的人时等于创建和维护软件全部人时的1/2。这比你所预想的成本高得多。



写得满满当当的故事卡片

需求谬论

人们最常犯的错误就是把故事卡当作了3x5英寸的需求文档。我把这种现象称作“需求谬论”。把故事卡当作需求可烂透了！小小纸片铺天盖地，故事卡变成了苦海。惨不忍睹。

你要是担心丢掉故事卡，那你就已经进入苦海了。就像我刚说过的那样，如果你想把卡片放到数据库里面去，可就过头了。另一种进入苦海的标志是卡上爬满了蚂蚁般大小的字体；还有一种标志是开始想用更大的卡片。

> **任何没有经过自动化测试的程序特性都可以被视作不存在……客户编写功能测试，他们对程序操作的自信也可以变成程序的一部分。**
>
> **——《极限编程精解》，57页**

说句公道话，图中这张卡片是历史上第一个XP项目用过的。但是在我找到它的那个页面上用大大的红字写着：“别这么干！”

如果不在故事卡上写需求，那这些卡片将去往何处呢？其实，从一开始XP就给出了答案。但大多数人都忘了。

客户验收测试

XP希望通过客户测试跟踪故事细节。但是因为它很困难，所以人们基本都不这么干。我们有xUnit帮助开发人员进行测试，进行测试驱动开发，但是客户测试就没有同样方便的工具了。

有些团队创造出来了他们自己的工具，但是在客户测试中还有第二点是更难企及的。我们希望客户编写他们自己的测试，所以我们不但要有一款用于客户测试的工具，而且这款工具还要能方便客户理解和使用——这里说的客户是产品负责人和他的团队。

在2002年，Ward Cunningham创造出来了这样一个工具，叫做Fit。Fit可以用来帮助团队在故事细节上进行写作。它也是个测试工具。但我更关注协作方面的功能，因为我觉得人们太关注Fit的测试了。Fit不是让你用来写完需求以后就交给开发人员的，它是让你用来跟开发人员一起工作，理解故事的全部细节，编写测试，编写能够持久的规范，作为将来参考用。

刚才这段话基本上都是根据我对Fit文档的理解总结出来的，尤其是这两篇文章：“Fit简介 (http://fit.c2.com/wiki.cgi?IntroductionToFit)”和“Fit工作流 (http://fit.c2.com/wiki.cgi?FitWorkflow)”。这些文档可以作为本次演讲的补充。

结论

总而言之，你需要一位富有远见卓识的产品经理在团队里工作。如果有了这么一个人，你就不需要预先定义所有的需求细节：你可以等到最后负责时刻再做。如果一个发布尚有三个月之远，你就只需要一个宏观把握而已；在实现某个特性的六个星期前，用卡片来对特性做简单描述就够了——这时候你需要用故事来做计划，开始进行实现以后，就要描述具体细节了。Fit是个很棒的工具，你可以用文字处理软件来编写客户验收测试，然后跟测试的某些文字描述合并。Fit文档还可以用作产品规范。■

作者简介

James Shore是一名独立咨询师，他的方向是帮助软件团队追求卓越的实践能力。他是《Art of Agile Development》一书的作者，该书对初、中级敏捷实践者提供了面面俱到的指导。在他的站点jamesshore.com上有更多作品。

作者简介

李剑是InfoQ中文站(www.infoq.com/cn)敏捷社区的首席编辑，Ethos(宇思信德)资深工程师。他的译作包括《深入浅出Struts2》和《硝烟中的Scrum和XP》，二者均为InfoQ中文站迷你书。

■ 责任编辑：郑柯 (zhengke@csdn.net)