敏捷开发产品管理系列之一：序言及设立迭代目标

敏捷开发产品互联网活动工作

目录(?)[+]

本文是敏捷开发产品管理系列的第一篇。（序言及设立迭代目标，产品版本规划，产品用户群规划，新产品研发，预估会议，Product Servant，Product Owner团队，产品线管理）

序言

之前的“敏捷开发用户故事系列”已经提到了微观层面的需求管理问题。由于敏捷开发的提出者和实践者主要是开发团队及其领导，因此一般较少提及产品的整体规划、商业目标这些内容。

本系列汇集了本人在做产品管理的时候的一些心得，以及在与不同企业交流、做互联网软件分析的时候的一些所得，与大家分享。

本系列的顺序整体由微观到宏观排序，拟包括设立迭代目标，产品版本规划，新产品研发，Product Owner团队，产品线管理等话题。

为何设立迭代目标

之前笔者的博客中曾经两次提到关于迭代目标设立的话题。

基于版本规划的迭代目标

一次是关于“迭代期内无变更”，即由于产品应该预先设定商业目标和客户群，因此整体版本规划也应预先设定，进而可以分解出相对稳定的迭代目标。

若能完成上述工作，则迭代期内尽管有微弱的变更，但迭代的总目标不会发生大的变化，从而保证迭代期内整体开发内容的稳定。

基于故事群的迭代目标

第二次则是提到用户故事的组织时，引入了“故事群”的概念，即某个迭代不应该简单地开发“当前最重要的功能”，因为万一这些最重要的功能零散地分布在不同的业务模块中，开发者就要同时面临同步了解多个业务模块的困境，前来评审的客户也会有如瞎子摸象一般只能看到多个局部的一角。

相对容易开发的方式，是在一个迭代中，应该安排相关的一组故事，从而将开发和评审的焦点都集中在一起。这样开发出来的产品也不完整，但却相对完整地交付了一部分功能，比之零散功能还是更有价值。

上述两种方法，一种基于外部的商业目标，一种基于内部的开发方便性，但都指向相同的结果。

怎样设立迭代目标

会前准备

迭代目标是提前设立的，早于计划会，甚至早于Product Owner将有开发意向的Backlog条目计划到迭代中。它实际上在做产品版本规划的时候，就应该捎带着被完成了。

它常常是一句简单的描述，如“一个能登陆和显示故事列表的版本”。

在迭代计划会之前，Product Owner会审视这个迭代的目标，从而决定将哪些故事置于本迭代中开发。

事实上，若已经设定了多个迭代的目标，Product Owner应该会同开发团队骨干，为未来2～3个迭代大致分配所需的用户故事，而不是赶在当前迭代前才匆匆分配。这个活动有利于开发团队骨干提前了解未来，从而在架构上做一些准备。

长期存在的一个难以平衡的问题是：若在架构上做了过多的准备，若中间发生变更乃至放弃某些功能，这些准备会浪费；若架构上准备不足，不断迎来新功能又会导致过多的“重构”发生，也会造成浪费。为多个迭代设立目标可以有效地帮助开发团队做出切合实际的架构准备，将浪费降低到最低点。

会后核对

在计划会上讲解故事、估算故事后，事情常常有所变动。

这时候要从已经计划要开发的故事总结一下看看，是否与原来设定的目标相符。

开发中跟踪

在日常开发中Product　Owner常常提出变更，若变更符合目标甚至能更好地达成目标，则应该被积极地接纳；若背离了目标，则应该缓做或重新考虑。

若“被激励”的程序员有了奇思妙想的时候，团队同样应该冷静地思考，做出判断。

“拥抱变化”与“迭代期内无变更”的矛盾其实向我们揭示了敏捷开发中的一个重要原则：变化的是路径，不变的是目标。

为每个迭代设立目标，非常好地让我们能够遵循这一点。

敏捷开发产品管理系列之二：产品版本规划

本文是一篇旧文，原名为《“迭代期内无变更”与敏捷开发产品版本规划》，因符合本系列内容，做相应修改后重新编排发出。

迭代期间无变更？

支持派说：对，如果经常变，我们怎么开发啊。

反对派说：不对，敏捷开发不能上来就确认了需求，要的就是在开发中逐步了解需求，怎么可能不变呢。

只在开发层面，这个问题无解。让我们站在产品版本规划的高度来看这个问题。

基于商业目标的产品版本规划

下个产品版本（或下个迭代）中到底应该有什么功能？最重要的功能？最基础的功能？当前可能实现的功能？已经弄清楚的功能？

这些角度都是基于技术活动而非市场目标来制定的，都有其局限性。

其实，每个产品的版本都是企业的一步棋：在某个时间，推出某些功能，满足某些需求，获取某些客户，打败某些对手，取代某些产品。

若认同了这一点，则早在产品版本规划的时候，就应该确认此版本中应该大致包含哪些功能，而非到迭代计划会议或迭代中才会确认，更不会在迭代中间发生变化。

这样看来，“迭代期间无变更”指的是：“不应该到迭代开发已经开始了还没明确要开发什么功能”（What问题）；而不是：“应该在迭代前把需求弄明确，一旦开发了就别改动了”（How问题）。

产品版本规划步骤图

产品立项 ------------------------------------------- 在这个时候大致规划出路线图，走多远，多久，走到哪里

V1.0 --------------------------------------- 在这个时候明确规划处这个版本要做哪些功能（未必到达故事点的粒度）

Sprint1计划会 --------------------------------- 在这个时候达到故事点的粒度，且从技术角度思考可以先做什么后做什么

日常工作 ----------------------------- 细化做成什么样子，随时可以变，但基本不会大量扔掉或换掉什么功能了

Sprint2计划会

……

Sprint Release ----------------------- 在这个时候，无论技术顺序的先后，所有V1.0的功能都做完了

V2.0 --------------------------------------- 根据市场反馈，调整产品路线图

V3.0 --------------------------------------- 继续

从这一点上，敏捷产品版本规划的目标与设定迭代目标的初衷相同：在“事先计划防止返工”与“随机应变防止想太多没用上”之间找到平衡，降低浪费。

敏捷开发产品管理系列之三：产品用户群规划

产品敏捷开发工作互联网windows游戏

目录(?)[+]

上周在培训做“用户故事的用户建模”练习的时候，就有人提出一个疑问：这么短的时间里边，能定义好用户群和用户群分类吗？答案是：不能。

用户群的规划是产品概念期就应该完成的工作，它是一个产品管理工作，而非需求管理工作。

用户群定位

这里仅就自己所从事的互联网行业提出一些简单看法。这些看法与本人在做的产品密切相关，因此在别处可能会不适用，仅作参考。

我们整体把用户分为人气用户和付费用户两种。

这个与互联网通行的分类方法很接近，相信很多企业给产品用户的分类也是如此。那有什么可谈的呢？因为若不把用户群定位的想法仔细写出来，产品做起来容易走样。

在传统行业乃至传统软件行业里边，用户基本上就一种：付费用户。你不买我的产品，就不会用上我的产品，我们基本上没有任何关系。这个分类的弊端在于：所有风险都压在用户身上，付费后一旦证明选择是错误的，后果自付。这件事情造成几个问题：

1. 用户付费前犹豫不决，售前和销售成本很高，造成销售和售前人员甚至多于开发人员

2. 用户如果不满意，因其损失很大，会产生很大的负面作用

所以（限于商业秘密写得很模糊）：

1. 人气用户负责做免费的市场工作，消除售前和销售成本。

这要求在给人气用户准备的免费版本里边做好一些工作：免费，简单，上手快，能互动点评，能传播，能试探性使用付费产品……

2. 人气用户中的一部分将成长为付费用户，用于提供销售额。

这要求给付费用户的的版本里边做好另外一些工作：从免费版直接升级，包含具有财务决策权者所用的功能（升级购买动机），快速的合同和付费渠道……

太细节的内容不方便说，这里点评一下“快速的合同和付费渠道”，其目的是有效降低售前成本。

“快速”有多快？点一个按钮，网上付账，电子格式化合同，自动发送许可，结束。

“客户连人都没见到，肯付费吗？”如果不肯，就说明我们的免费版本做得不够好，或他作为人气客户期间还没有获得足够的信心。我们会在免费版本中做得更好，消除其疑虑，而不是动用销售人员。

这一点同时也就是规划好了“免费版本”到底要做成什么样子，它不再只是一个“功能相对简单，许可受限”的产品，而是赋予其“完成售前信心建立”的使命；这两个目标下产品的定位截然不同。

再说就到核心商业机密了，大家自悟吧，呵呵。

不同阶段用户群的变化

产品在长达10年的生命周期中，不同阶段会接触到不同的用户，会产生另外一个维度的划分。大致如此：

1. 粉丝用户

无限认同产品理念乃至认识产品开发者的用户。介意产品风格与内涵，不介意易用性。

会提出大量意见甚至多数意见都是他们提出的，但是其追求的并非大众化的功能，而是具有很强的价值观取向的功能。

对待这些用户在早期应积极响应，形成影响力；中期应审慎对待其需求，防止产品走向极端。

2. 试探性用户

有新东西就会尝试，但是本着实用主义精神，因而缺少耐心的用户。

因此中期产品的易用性和易上手性非常重要，否则过不了试用关。

3. 大众用户

不爱尝试，受到以往用户的影响多于自身的体验感受，纯实用主义者。

易用性和标准性非常重要。所谓标准性，就是产品具有相对一致的用法，因此很容易搜索到，很容易培训大量用户。

大众用户的数量庞大，必须利用粉丝、试探性用户形成的知识对其进行市场、销售、支持活动。

这句话什么意思呢？去游戏网站转转就知道了，各大游戏网站的BBS区都不是由游戏公司的人负责回答问题的，而是广泛发动粉丝、试探性用户对大众用户提供支持。

因此末期产品的产品本身与其外围支撑系统浑然一体，比如QQ，比如360，点着点着，你已经分不清楚到底自己在产品里边，还是产品的支撑系统（就是老观念中的“帮助文档”）中了。

非互联网行业也大致存在类似的过程，如微软为粉丝用户做了DOS（相当难以普及，因为操作复杂，但绝对有人用），为试探性用户做了Windows3.1/3.2（非常容易上手），为大众用户做了Windows95（从此之后的界面就没怎么大的改动过），完成了其“每个人电脑中安装着MS操作系统”的商业计划；但随着Win95/98装机量上升后，后面的产品就越来越不知所云了。因为“每个人……”的用户群计划已经完成，而新的用户群计划是什么呢？没有，所以才有Win7大战XP这样的内战出现。

配套商业模式

为配合这样一个产品的规划，很多配套的商业模式也要随之产生。

这里不方便说太多，但无外乎：免费，长尾，集合器，生产者与消费者合二为一，半成品思维，无重经济……这些常见的互联网策略。当然要把这些内容落实到实际工作，每个企业都要付出很大的努力。

抛开用户群谈产品规划是空谈，用户群规划是敏捷产品管理的起点。

用户群定位的方法很多，有空间方面的，也有时间方面的。但其目的只有一个：理解应该做出怎样的产品及版本，才能赢得这些用户，达到商业目的。

敏捷开发产品管理系列之四：新产品研发

目录(?)[+]

这里所指的新产品研发，不是指自己企业的新产品，而是特指那种在行业中初创，前途不明，尚需市场检验的新产品。敏捷开发可以在很大程度上帮助这种产品的开发过程。

新产品的第一要务

策划新产品的第一要务是：谁会买这种产品？为什么？

开发新产品的第一要务则是：它与以往产品的核心差异是什么？

这个听起来不难理解，但是做起来有困难。因为一般产品开发往往是先做“最重要的功能，最基础的功能，最影响架构的功能”，这很容易在很久以后，才能看到产品的核心差异。

因此，虽然不可能完全脱离重要性、基础性、架构性的制约，但仍然应该常记：验证与市场上以往产品的核心差异是第一要务。

新产品研发

如何快速体现核心差异（一般是创新性的价值观），是新产品研发的中心。

具体做法很多，下面举一个例子。

曾经有一家游戏企业，希望能用“广种薄收”的方法来做新游戏。但是发现每个游戏上来都要做可视化、用户登录、商店、NPC、场景、帮派……这些基本功能，所以经常游戏开发开始很久了，还看不出游戏“好不好玩”这个最重要的核心。这就极大地制约了人才、资金的流动性。

后来他们决定：开发一个基本平台完成上述基本功能，任何团队拿到这个平台，直接在其上开发“核心玩法”，在规定时间点上公司领导会评审游戏的可玩性，来决定其去留。这就是一个让团队将开发重点从基础功能转移到核心玩法。

再举一个我自己的例子。

我们正在开发的产品在市面上已经有很多了，因此尽管从敏捷开发的角度看应该尽快做一个“可工作”的产品出来，但我们实际过程却相反。

我们的确有“可工作”的产品，但只限于部分体现核心价值的功能点上。单单靠这些功能点，整个产品其实跑不起来；但整个产品跑得起来不，并非我们评价产品成功的要素；反而是少数不连贯的功能，体现了我们产品的价值。因此在早期的开发工作中，整个产品其实不能完整地“工作起来”，但是那几个能体现核心差异的功能，却能帮我们验证是否值得完成整个产品。

敏捷开发的帮助

敏捷开发正好可以利用优先级排序、评审会、迭代交付等实践，帮助完成上述步骤。

尤其是迭代交付。即使只有3个月的初创期，也不要认为时间比较短可以在前期作出完整的功能或设计。迭代开发能够保证在第3个月的判断点来临之前，自己与仲裁者能有多次沟通的机会，仍能改进产品的方向；而且在真正的时间点上，一定有可交付的功能，而不是一堆做了一半的文档。

“只有一堆文档”虽然未必导致老板直接把项目砍掉，但至少会让他很为难。不如用迭代给他一些“看上去可以，但仍有待判断”的功能，以便获取更多机会。

敏捷开发产品管理系列之五：预估会议

目录(?)[+]

粗估会议是一个较少使用的敏捷实践，但其作用还是很明显的。

Why

Scrum里边，有两个关于需求的比较头疼的问题。

一个是PO不太懂技术，不知道故事大约需要多久才能完成。为什么PO要知道？因为如果划分的颗粒度不好，会导致有的故事太大或太小；而且如果不知道故事的大小，就比较难大致预测未来的几个迭代能做完哪些故事，版本计划不好做。

二个是如果只依赖短暂的Sprint Planning Meeing，往往团队对故事的理解不够深刻。人的思维方式往往是分为两种的：一种是集中思维，就是现场的讲解问答；另一种则是分散的思维，就是“回去以后越想越不对劲”的那种，后者如何利用呢？

开一个预估会议把。

When

有一家游戏公司，会在每次Sprint Planning Meeting前一天，先把整个迭代要开发的大致内容讲解一下。这次讲解，未必是按照故事条目来的，更多的是整体的介绍。因为游戏的需求相对复杂，而且关联关系很重，所以有必要整体把握以下。这很像拍摄电影，导演会把下一个场景整体介绍一下，才会分解镜头拍摄，否则整体感和连贯感会很差。

有一个日本企业，则是在30天迭代的第10天左右，开下一个迭代的预估会（或称为预测会也行），整体就是提前20天介绍一下下个月有什么事情要做。这个目的，则是让队员在剩下的20天里边，都会潜移默化地思考下个月的事情；另外就是在开发本月内容的时候，如果成本很低，会提前做一些准备工作（比如预留点接口。“过度设计”很容易造成浪费应该避免，但是为20天后的工作内容做准备，则极少会浪费，所以值得）。

How

在预估会上，PO给大家讲解整体需求，和他对下个迭代的期望，以及他提前拆分出来的几大块或几小块内容（看工作习惯了）。

团队会分析这些内容的关联性、实际工作内容、工作量规模等内容，必要时拆分故事，描述故事的依赖性，并进一步帮助PO形成条目化的用户故事。

PO在形成有工作量预估的用户故事后，会重新计算和规划下一个迭代的工作内容，让其能更完整地组成一个发布。

当然这次估算不一定准确，力求做到条目化和拆分即可，准确性由参会者在直觉上保证都可以，毕竟在计划会上还会细化。

Who

有时候整个团队参加，尤其那些对完整性要求比较高的产品，比如游戏。

有时候只让小组长或骨干参加，他们会把参会的内容传达给自己的队员；更少的参会人员，可以降低对其他无关人员的工作量占用影响。

这取决于产品和团队的情况，试着开几次会议，自然会发现怎样才是合适的。

TODO

PO开完会议后，可能有这样一些工作要做：

1. 记录下来最终被条目化的故事。

2. 在计划会之前，把会议上似是而非的一些内容敲定。

3. 根据故事的大致规模，规划下一个迭代的内容。应本着相对完整内聚的原则，组织成一个用户能真正拿到并完整工作的产品。

4. 按照MoSCoW排序出下个迭代的内容。（MoSCoW请参考http://blog.csdn.net/cheny\_com/article/details/6358456 ）

敏捷开发产品管理系列之六：Product Servant

**马与马车夫的故事**

这是心理学上的一个比喻。

马拉着车向前走，它说：控制车去哪里的是我，我走，车就走；我停，车就停；我转，车就转。

马车夫笑了，他说：控制车去哪里的是我，我让车走，车就走；我让车停，车就停；我让车转，车就转。

后座的客人笑了，他说：难道你不是以送客人回家为生？

当我们成为Product Owner的时候，一个极其危险的旅程开始了。

**产品的生命周期**

几乎没有产品，在开始赢得市场之后，乃至如日中天后，都会走下坡路。

如果一个产品的基础架构及其难以调整，导致技术老化，这还可以理解。

但是很多消费电子类产品，技术的更新换代很快，但是某些产品还是很快老化。

如果一个、两个产品偶然做错了方向，导致产品概念老化，这也可以理解。

但是如果很多产品，甚至出现一家公司的所有产品都“老化”而不受欢迎的现象，就值得深思了。

**在创业之初，为了能从众多既有的先辈们中间杀出一条新路，多数创业者都能保持谦虚的心态，从市场对先辈产品的或多或少的抱怨中，找到一个突破口**，进而创新或微创新出一个新产品，赢得客户的亲睐。

但一旦成为行业的领先者，情况就发生了转机。当前面还有先行者时，人们或可以选择老路追赶，或可以选择新路超越，还算是有路可循。

但自己成为先行者时，人们就很容易认为路就在自己脚下，自己向哪里走，大家就会向哪里跟随。

**进入后者的阶段，企业就较少倾听市场和客户的声音，而执着于自己的想法，并相信自己能够预见未来。**

**破解**

“等我有了钱，我一定做一款我认为世界上最好的XX。”这种愿望在初入职场的程序员心中都有，就何况企业的领导者了。

因此首先应该意识到自己一定会走偏，然后才能正视这个问题，才能找到答案。

**作为基层产品经理，应多了解客户和用户的业务，而避免闭门造车。**

即使是基层的产品经理，一般也有一定的阅历，这种情况下极容易产生“愚蠢的客户”的概念，就是认为自己的想法比客户好，因此总想用自己的想法代替客户的想法。

很久以前开发一个滑雪场的计费系统，就对客户的操作模式很不认同。比如租赁滑雪服只登记颜色，而没有编号；颜色不用手敲也不用下拉菜单选择，而是用键盘……等等。后来才知道操作这些软件的，是附近的农民，在大雪封山无地可种的时候来雪场工作四个月。任何ID、编号、输入法、下拉菜单、快捷键……乃至我们认为最好用的鼠标，都是令人头疼的因素。

之后我们亲临现场与雪场工作人员讨论，甚至到附近另外两个滑雪场滑了几次雪，来了解整个过程中到底会出现哪些问题。

**作为高层的产品总监，要谦虚地观察和借鉴小型竞争对手的举措，以防止自己的老化。**

人，乃至公司会老化，已经是一个不争的事实。

很多大型公司都投入了无数资金和人力进行创新，但反观过去20年的主要IT创新，没有几个是业界领先者做出的，一半是大学生+车库创造的，一半是生死一线+回光返照做出的。

网上盛传的史蒂夫鲍尔默的笑话：第一次他拿到iPhone，说：“这台手机充完电连一天都用不到，谁会买。”；第二次他拿到iPad：“这东西没有商用价值，谁会买。”这些人都是当世英雄，自然不是愚蠢的人。为什么会说出这些话来？

人老了，观点就执着了，尤其是从胜利中走过来的人，尤其如此。一个人在60岁的时候，让他承认说“我之前有生之年的思想，现在已经过时了，需要重新建立一套思想”是极其困难的（这是一本励志书上的内容，作者因此教导我们要尊重和理解老年人的“老顽固”心理）。

**若要让企业避免随自己老去，作为高层的领导，不能认为自己乃至自己的企业会永远正确，要虚心地向新企业、小企业学习，这些企业的年轻将驱使他们只有创新才能生存，这种创新不是钱或人力能换来的。**

**Product Servant vs. Product Owner**

因此Product Owner，实际上是一个很令人走火入魔的名词。

之所以敏捷开发使用这个词，原因是让开发团队意识到自己不是产品的Owner，而另有其人，比如产品经理，他们才是Product  Owner。

但作为产品经理，也要意识到其实自己也不是产品的Owner，仍然另有其人，就是客户。

而甚至是客户，如果不是手机、游戏这些消费软件，而是OA、MISS、银行、电信这些再为客户的客户服务的软件，Product Owner也另有其人。

**而当Product Owner变成Product Servant，才可能真正走到用户之前想问题，而又不至于偏离实际需求太远。**

 敏捷开发产品管理系列之七：Product Owner团队

## 目的

在之前的《Product Servant》一篇中曾经提到，作为产品经理或产品总监，都应该有自己的方式来根据市场和用户情况来管理产品的走向，其中前者更倾向于具体的功能，而后者则更倾向于市场方向的竞争力；前者要求细节，后者要求高度。那么，这两个人到底谁是传统意义上的Product Owner呢？

另一个常常被问到的问题是：“我们只有一个产品经理，而有20多个开发人员，这一个产品经理能忙得过来吗？”

再有一个问题则是：“PO只在迭代开发会上沟通需求，还是在迭代开发期间仍然与团队在一起？”

还有一个问题则是：“PO在最后的评审会上才看到结果，万一发现问题需要改正，是否会为时过晚？”

又有一个问题则是：“可以让客户代表参加计划、评审活动吗？”

……

这些诸多问题，都指向一个答案：“**一个PO很难完成其所应履行的各种职责，我们需要一个PO团队。**”

## 结构

PO团队整体上是一个上下分级、内外疏密的团队。

**所谓上下分级，就是说一定要有产品总监层面的人员参加，以便把控整个产品的走向。这种把控的结果，是能按照商业步调形成大的版本计划。**

若失去这种大的方向，就很难真正“按优先级排序”，因为真正的优先级，来自于对盈利性产品的持续盈利能力、新产品的价值判断发展方向等商业目标的追求，而不是技术上的优先级。

在总监层面的工作完成后，产品经理会在具体版本、计划、需求开发、评审、发布等工作中将商业目标落实到开发工作中。从时间上，保证产品进度符合商业步调，从空间上，保证产品需求符合目标客户群。

在大型的PO团队如游戏的策划团队（常常占总人数的1/4）中，还会有三层结构：主策划（产品总监）-策划组长（产品经理）-策划人员（产品助理），负责一小部分故事的编写与跟进（跟进活动见下文）。

**所谓内外疏密，则是在开发过程中由客户、市场、销售、产品、开发、售后等综合角度审视产品**。

比如游戏公司常常邀请运营部门或发行商参与产品的管理，消费电子则会邀请分销商、售后部门等，目的是提供第一手的反馈。

产品研发团队的代表也常常参加PO团队，来帮助PO团队把握产品演进的技术路线。研发团队代表还会从整个商业路线图中勾勒出技术路线图，来判断是否以及在何种程度上“为未来做准备”。（在《智慧敏捷系列》中曾经提到，准备太多会浪费；准备太少会返工。）

**产品经理在整个过程中扮演穿针引线的作用，即作为各方的Servant，向上提供决策依据，向下提供目标指引，向外提供产品支持，向内提供用户需求。**

## 活动

除了常见的需求优先级排序、计划会讲解故事、评审会评审需求之外，整个PO团队还有很多工作，下面按照工作的大小、层次列举一下。

* 产品初期
  + 产品总监：设定商业目标和路线图
* 日常工作
  + 产品经理：制定发布计划，形成和描述故事
  + 研发团队的代表：制定技术路线图
* 迭代前
  + 产品经理：优先级排序，选择下个迭代的待开发列表（Willing List），选择故事群，制定迭代目标
  + 产品经理/研发团队的代表：预估故事，协助拆分故事
* 迭代中
  + 产品经理/产品助理：负责细化需求，跟进需求的完成情况（渐进式评审，见下）
* 评审会
  + 产品总监/产品经理：评审需求完成情况，提出意见，重新排序

某些PO团队参与开发过程的活动，请参考：http://blog.csdn.net/cheny\_com/article/details/6905291

## 渐进式评审

所谓**渐进式评审**，又叫跟进人制度，是为了防止迭代最后一天的评审会上发现了问题，由于第二天就想发布了所以来不及改正，因而将评审改为每一个故事完成，都进行一次评审，若需要改正则提前进行。

一般需要以下实践配合：

1. PO团队人数较多，且有层次，能为每个故事设定跟进人，负责解释和验收这个故事。

2. 配合[MoSCoW方法](http://blog.csdn.net/cheny_com/article/details/6358456)，按优先级逐个完成故事，若M、S类型的故事评审后需要改进，则可能牺牲后面的W类型工作。

3. 中期评审会。有些团队在为期30天的迭代的第20天左右，会开一个预评审会，以完成2中提到的工作。

敏捷开发产品管理系列之八：基于业务设计技术架构（兼谈12306性能问题）

在产品开发中，常常遇到产品性能问题，这些性能问题会很大程度上影响到产品的架构。

但解决这些性能问题，切莫认为只是技术人员的事情，产品经理和产品总监也要参与其中，甚至是业务人员（销售、售前）。

下面以12306的售票问题为例，来做一个完整的说明。

本文的目的，不是说技术性优化不必要，而是说作为开发人员不要闷头只想技术，而作为产品经理不要把所有“技术”问题推给开发人员，这一点很重要。

技术方案的局限性

12306为什么崩溃了？

原因众说纷纭，解决方案也众说纷纭。

到网上一搜“12306 性能”http://www.baidu.com/s?wd=12306+%D0%D4%C4%DC&rsv\_bp=0&rsv\_spt=3&inputT=8540支招者不计其数，但多数集中在技术方面。

本人对数据库一项所知甚少，也不懂如何优化，但下面我们从业务的角度看看有没有什么解决方法。

先看看这个方案：为何不上10倍的服务器？

很多人提出，如果上10倍的服务器（或10倍的内存/硬盘/……），可能就解决了拥堵问题。

实际上我也想过是否可以上10倍的火车，解决中国的春运问题。答案是不能。

因为任何能够满足春运数量的火车，在非春运的时候都会变成很大的负担，只能停在什么地方风吹雨淋等待下一个春运到来。

所以，突发性是核心矛盾，无论用技术方法，解决突发性是主要矛盾。

突发性的放大

除了很多人在这段时间买票之外，有一个正反馈过程加重了突发性，那就是买不到票。

可以说，访问人数无论上升还是下降，只要还有票，多数人都只会登录此网站一次，性能问题至少还是线性的。

但如果买不到票，或买不到某个车次的票，人们就会突然多次访问网站，性能问题就变成非线性的了。比如有一个人就刷新200多次，终于购票成功。

把200变回1，这不是一个简单地利用技术能解决的问题。

在某个攻略中也提到，由于人数太多，登录都需要20～30次才能完成。这些非线性因素，乘上本来就增加的人数，难免崩溃。

从业务角度思考问题

先胡写几个方案看看，先假设我不会编程。

1. 把提前售票时间从10天改为20天

“什么？”是的，这个傻方法差不多可以降低服务器负载50%。

2. 设计一个排队系统，完成登录

以前玩游戏的时候，经常因为部分服务器崩溃而无法登陆，不过，这时候都会出现一个排队系统：“您正在排队登录，前面还有1000人……900人……800人……（别乱动键盘啊，快到你了）”

这个是用来解决雪上加霜的突发性放大问题的。

我相信12306肯定为30亿人次的春运做过准备，只是没为6000人次的春运做准备，排队系统可以把人数重新变回30亿。

3. 设计一个排队系统，完成预订票

进去了，还是没有票，怎么办？每天抽空就上来看看，然后重新登录……刷新……又是一个突发性放大因素。

如果有一个排队系统，你按优先级排列上自己想买的几种票，甚至直接说“某月日，哪到哪，无论车次，有票就给我”，在家等着退票，或者偶然发出临时车次吧。

如果是我，我还会做个短信服务，如果买到了就短信通知；如果还在排队……如果你愿意接收，每排名向前10%，可以再发个短信；你耐不住性子想查询一下，也可以反向发个短信来，实时查询。

不过，短信要付费的，1毛一条，平价的。我听说短信分账是2：8开，电信才拿2分钱，剩下8分归铁道部，不知道现在是否还是这个规矩。

一个春运下来短信还能赚几亿（应该完胜CCTV的春晚），而且客户还挺高兴，毕竟这几毛钱解决了大问题了，免去半夜爬起来/请假/寒风中排队。撇开这些不说，一台台式机开机3小时就是一度电，6毛钱，管保3小时内你买不到票的；而现在能了（如果还有票）。

当然，如果铁道部愿意让利于民，免费发短信就更好了，不过如果能解决这个问题，相信实际上大家都会觉得让不让的都无所谓了；毕竟火车票10年没涨价，这些钱给人家发加班费吧。

另外这个排队系统还能把黄牛的刷票软件干掉，黄牛再多，也跟着十亿人民一起排队吧。

……

等等诸种业务方法，虽然不能解决有没有票的问题，但能解决购票的性能问题。

实际客户体验也要好得多，毕竟无论你怎么上服务器和优化技术，如果我还是上去200次，依旧耽误工作和生活，弄不好还的从黄牛党买票（他们有专业刷票软件，从性能优化的收益远超过我们）。

从业务角度思考技术架构

正题反而很简单了，如果要做技术架构，先要了解业务架构。

这一点要说服产品人员和业务人员参加进来，在http://blog.csdn.net/cheny\_com/article/details/7220270里边的案例中提及了如何让商务人员进行架构设计的问题。

《火星人开发纪实：敏捷开发一千零一夜》序言

本文是《火星人》系列的子系列，将分期向大家分享火星人敏捷开发管理工具的开发和管理实践。

一直以来，敏捷开发长期受困于各种名词、术语的堆叠、罗列、解释，而较少出现原创和实践分享过程。而敏捷实际上本来只把自己作为一个起点，需要大家的丰富和扩展。这可能与中国的软件业发展长期落后于国际所致，以及在PMP、CMMI推广中所养成的重标准、轻实践的的情况有关。

本子系列会与大家分享我们自己的开发和管理实践，它们可能不完美，也不是终极实践（因为我们未来会做地更好），但却是在时间、人员、市场、产品、团队……众多因素的限制下的真是实践。

每一期中，除了实践本身之外，我们都会尽量分享关于这个实践我们的考虑过程、未来设想，以便帮助大家思考和实践出自己的敏捷开发来。

本文已经收录于《敏捷开发案例集》，有志于提交和发表自己案例的读者，可致信wanglijie9057@gmail.com，或申请加入QQ群：173709637。注意这是一个内部讨论群，仅对潜在的案例提供者开放。

序言

这是一个真实的故事。

这是一个只会在世界上发生一次的故事。

所以，它揭示的是真理————————在现实世界中的一次闪现。

去年在业余时间开始了一个产品研发项目，开始的时候一个人隔三差五地抽空开发，后来投入了比较常规的时间进行开发，最后又有一个兄弟加盟两个人一起开发。

这个产品，就是“火星人”，一个敏捷开发管理工具。既然是敏捷开发管理工具，自然就应该按照敏捷开发的规矩办事：用户故事，迭代计划，每日立会，自动化测试……一个都不能少，对不对？然而在开始了不久以后，我们发现问题没有这么简单。

下面的故事，就是摘选了其中围绕用户故事相关的心得体会，以后还会有一些关于测试、重构、迭代规划、团队管理等方面的心得。

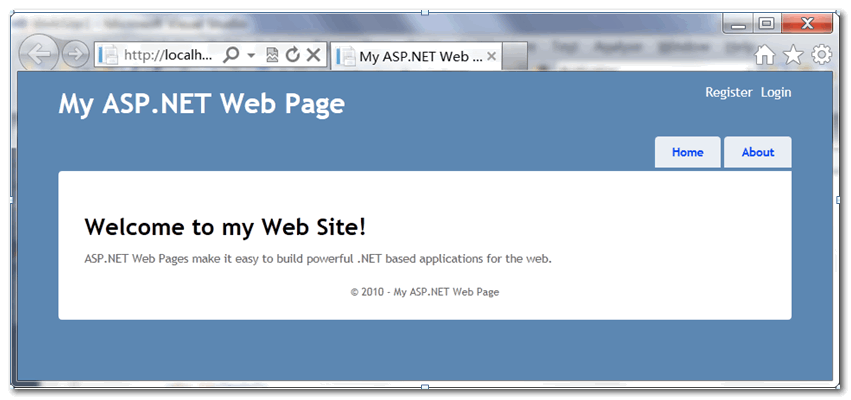
由于我们的前后人数变化、实际生产率、开发日程很特殊，下面的时间点做了一些加工，以利于可以像正常项目一样理解。

免责声明：本系列文章涉及的若干中间产品截图，本文作者也认为非常丑陋。但为了保持纪实性，仍然保留这些在实际产品中已经不复存在的历史资料。请在其他正式火星人产品文章的指导下进行阅读。

# 《火星人开发纪实：敏捷开发一千零一夜》第一个月：一个产品的诞生

## 第一个月：一个产品的诞生

没有国王，没有宰相，没有能讲故事的王后，也没有需求文档，开发就这样开始了：



## 为何不先写需求文档？

**因为敏捷开发不写文档！不是的。**

策划这个产品的时候，有一个大愿：就是用这个产品管理这个产品自己的研发。虽然这不等于没有“长得像”Word的文档的需求，但是我们仍然想尽量只用产品本身的功能来写需求。

所在在项目开始的那天，我们手里（确切说是脑海里）只有一个商业计划，外加这个产品的大致功能列表。

## 那怎么知道要开发什么功能？

         至少在开始的时候，这个问题不很重要。

         因为敏捷工具已经存在了至少10年了，里边到底应该有什么功能，在前辈们的网站上一搜，都能找到。但是，完成这些功能，乃至超越这些功能，都不是我们的目的。因为在软件界，一个已经做了10年的市场，本应是一个没有投资价值的市场。

**除非，发现了全新的价值观。**

## 一个产品是怎样诞生的？

限于商业机密，这里就不把我们当时发现的新价值观和商业策略写下来了。下面泛泛地讲一下一个产品诞生时应该发生的事情。

几乎所有产品，在开发出来前，都有类似的产品存在。

几乎所有产品，在开发出来后，都有类似的产品跟进。

几乎所有新公司，都比之前的前辈小，不可能追上前辈。

几乎所有老公司，都比之后的后辈大，很容易转向踩死后辈。

**若不想被前面的骆驼拖死，又不想被后面的大象踩死，就要走一条骆驼或大象都不会走的路。**这种路很少很少，以至于很多人花费很多年才能发现一条，但是幸运的是它不但被我们找到了，似乎还已经装好了路灯。

早期的开发，就是要证明这条新路可走，而不是把骆驼的路重新走一遍。**所以在最一开始，我们并不需要一个完善的功能列表，而是要那个核心价值，以及一些能体现那个核心价值的功能即可**。

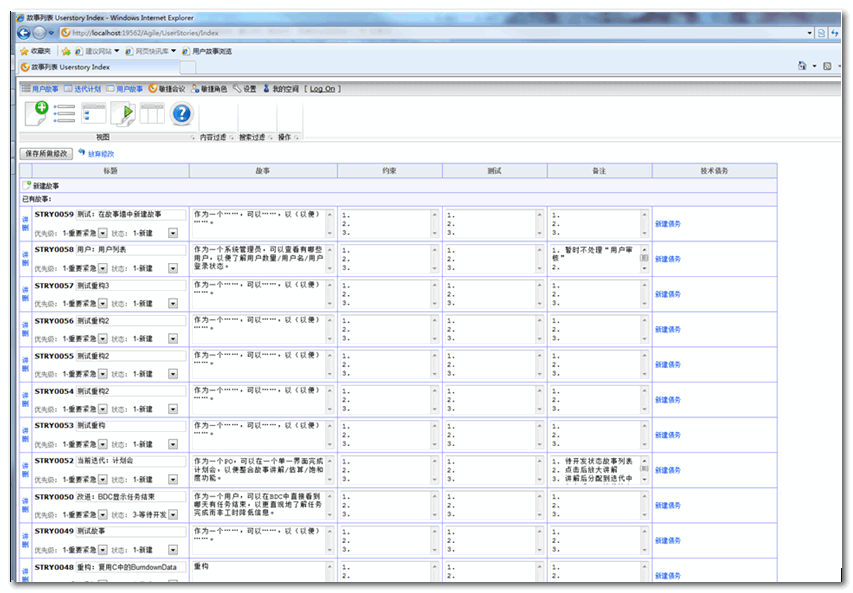
这是新产品研发的核心原则。

## 那先开发什么呢？

         我们当时的选择，是“用户故事显示”功能。因为这是第一个我们会用到的功能。新增、编辑、删除，则直接在数据库里边做。由于选择了ASP.net、MVC2.0（现在用3.0）、LINQ等一票先进技术，第一周，故事就安然地呈现在屏幕上了。第一个愿望算是实现了。

又经过一个月的时间，增删改查都做好了，而且还做了一个集群编辑的界面，几十个故事一罗列，特宏伟。

这似乎预示着未来将是一帆风顺。



# 《火星人开发纪实：敏捷开发一千零一夜》第二个月：框架优先，还是故事群优先

## 先开发整个框架，还是先开发故事群？

         当用户故事的增删改查都还没做的时候，这个问题就被提了提出。

         最初曾经想：只做故事的显示功能，增删改都在数据库内直接操作，然后迅速进入“迭代计划”部分，把迭代计划大致弄出个样子来后，赶紧做燃烧图、故事板……

但是这个想法很快就被否定了，因为在此前做敏捷咨询的时候就曾经在用户项目里发现：**如果把一个产品的所有“最重要的功能”单独提出来，不但不会生成一个最小的可用功能集合，反而会生成一个血肉模糊的四不像**。

         所以就选择了把用户故事的功能尽量开发完整，再向前推进，这样的一个产品尽管缺少很多功能，但是已经有的功能却完整可用。我们还为**这样一组相对完整地进行分析和交付的功能称为“故事群”**。

但是在用户故事的增删改查全部完成的时候，这个问题又回来了：是继续开发用户故事排序、讲解、估算的界面，让“用户故事”相关的故事群更加丰满完整；还是转而开发迭代计划、燃烧图等其他模块的基本功能，先把架构搭起来？

架构优先还是故事群优先？这是个问题。

## 跟着感觉走

         还好不是第一天做项目了，不想坐在旁边让方法论打架；也不想依赖Google或者百度。我们的项目，我们做主。

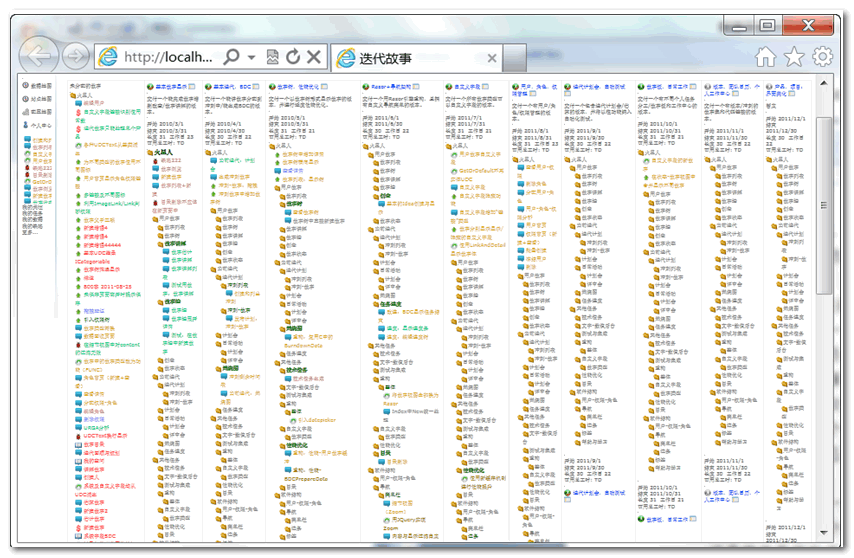
         终于找到一个标准，作为**判断的准则：那就是问自己，现在是更容易开发故事排序、讲解、估算的功能，还是更容易开发迭代与计划的功能？**思考的结果，答案是后者。

         现在回忆起来，这就有点像走夜路，隔一段点亮个蜡烛，很令人不放心；但只停在一个地方修建个大灯塔，就连走路的基本感觉都没有了。不过在蜡烛和灯塔之间，有一个巨大的落差，要找到的是那种像“隔一段有一个路灯”一样的折中点。

**每个产品每个项目，应该有自己不同的抉择点，不能固守成见。**

## 迭代，故事？故事，迭代？

第二个月完成后，产品变成这个样子（这张图中的目录结构，是第三个月产生的）：



最左侧的是没有分配到迭代中的故事，右边的几列则是迭代中的故事。说实话，很丑，但是我们已经可以尽情地创建故事，并把故事分配到不同迭代去了。在项目开始后的第二个月，我们终于拥有了自己最基本的项目计划。

迭代计划的出现，也令另外一个与架构和故事群相关的问题浮上水面：是基于故事制定计划，还是基于计划挑出故事？这有什么区别吗？有。

前者先写好用户故事，然后向迭代中分配，若一个迭代完成不了，则挤到下一个迭代完成；每个迭代根据内部分配的故事，总结出一个简短的名字。这个是很传统的迭代方法。

后者则是相反，先制定每个迭代的目标，并与故事群大致对应（比如前三个迭代目标是“基本故事显示”“基本的迭代计划和燃烧图”“故事树和性能优化”），然后向迭代中分配相关的故事。若一个迭代中的故事未能做完（一般是最次要的故事没有完成），则也不推迟到下一个迭代中完成，而是暂时封存。这样每个迭代尽量保持在做迭代目标的故事。封存的故事有系统管着呢，反正不会被遗忘，最后再说。

后来我们选择了后者。

原因是**如果说尝试每个迭代都把某个故事群“彻底做完”，几乎每次都会剩下点什么细枝末节的功能，扰乱下一迭代的完整性，是我们不想看到的。**

## 一波三折

         总结一下第二个月的工作，可谓一波三折。

         在优先完善框架，还是先就一个故事群交付的事情上，多次思考的结果都各有不同。以至于无法总结出到底框架优先，还是故事群优先。

         如果真要写一句总结那就是：**敏捷无有定法，每次都先找到自己的判断准则，然后跟着感觉走。**

敏捷开发一千零一问系列之二十六：如何进行优先级排序？

产品任务性能优化敏捷开发活动

目录(?)[+]

这是敏捷开发一千零一问系列的第二十六篇。（在这里提问，之一，之二，之三，问题总目录）

问题

如何进行优先级排序？具体故事的优先级，和版本规划的优先级之间有何关系？

分析

敏捷开发里边有很多地方需要多次进行优先级排序，本文将探讨其不同的应用场景，及其关系。

值得注意的一点是，敏捷开发中有无数的“自相似性”，比如估算，每年、每月乃至每天人们都在潜移默化地估计自己的任务；又如计划，也是每年每月每天都有成文或不成文的计划；而发布，也是每个故事自成单元，而又与其他的故事构成每月的交付，进而形成几个月的大版本交付……

由于这些自相似的活动颗粒度不同，一定不要认为只要做一次就可以了，也不要认为用一种方法做就可以了。要就具体活动思考这一活动的目标是什么，以及如何才能更好地达成这一目标。

方案

产品路线图级别的版本优先级排序

这个是最高级别的优先级排序，排序的预见性大约在3～5年左右；排序者一般是产品总监等高级管理者；其排序的对象一般不是用户故事，甚至不是功能模块，而是产品线和产品。

排序思路大致是：不同的用户群需要不同的功能 > 因此导致产品应该面向客户群开发 > 识别出当前最重要的客户群规划下一个产品 > 即使相同的客户群也需要逐步为其提供服务 > 根据客户的业务计划或用户的消费习惯制定产品的版本方案。

如果对此特别感兴趣，请参考文末的链接。

应该注意的是，往往公司缺少产品总监的职位，或者虽然有但是极少对中长期的产品规划进行计划，所以很多时候中层的产品经理也需要思考这个问题。

版本内部的迭代优先级排序

这个是中等级别的排序，排序的预见性大约在6～10个月左右；排序者一般是产品总监和产品经理；其排序的对象大约是模块及史诗故事级别。

史诗故事

先说说什么是史诗故事。敏捷开发里边定义史诗故事是一些“比较大”的故事，但没有定义其绝对的尺度。火星人认为，“业务数据”是一种很好的史诗故事尺度，比如我们说：一个产品（假设就是敏捷开发管理工具）要管理下面的数据，请问在半年的开发周期中，应该先开发哪些？

用户故事，优先级，用户故事树……迭代开发Sprint Backlog，迭代计划会……人员，角色，权限……部门，团队，小组……

我们就可以说，这样规划吧：

第一个月：用户故事，优先级，用户故事树……

第二个月：迭代开发Sprint Backlog，迭代计划会，任务分配Assignment……

第三个月：人员，角色，权限……

第四个月：部门，团队，小组……

……

第N个月：整合并发布一个版本

注意在这个尺度上，我们几乎不讨论具体的操作（比如对“用户”，应该有增删改查等操作），而只会讨论对哪些业务数据进行操作，这非常符合人们的工作习惯。

这种名词性质的业务数据，就可以当作合理的史诗故事来看待。

模块

而如果观察每个月的内容，会发现他们比较相关，比如“人员，角色，权限”这三个功能，如果把它们分散到不同的迭代中，很难单独开发。

这类强烈相关的业务数据，就可以称之为“模块”，划分的依据仍然是业务。

尽管很难让每个Sprint与一个模块一一对应，但应该建立这种意识，即每个迭代应该开发相对集中的一个或多个模块。

迭代排序依据

也就是怎么决定哪个模块或史诗故事优先开发。人们习惯性地会根据依赖关系来开发，但这不是一个好习惯。

比如，有人认为若系统连登录都不能，谈何继续工作，于是就是把“人员，角色，权限”放到首位，然后很可能会是“部门，团队，小组”，这样万事俱备之后，就能很好地开发用户故事、任务分配这些功能了。其实这种做法十分不妥。

第一个可能出现的问题是：由于还没有开发业务，所以其实很难说清楚需要什么样的人员、角色、权限管理办法才能支撑业务。

第二个问题更为突出：在第二个月完成的时候，其实我们针对业务什么都还没做，造成高层领导很难理解我们的进度，在投资、招人这些事情上会犹豫不决，因为我们已经完成的功能很难帮助他下结论。

在新产品研发的过程中第二个问题尤为突出，高层心急如焚急需知道产品的竞争力，而我们正不紧不慢地搭建底层架构。

所以正确的做法是：先从核心业务入手，进行优先级排序。

没有人员、角色、权限，那就大可让每个人无需登录就可以管理用户故事、优先级、用户故事树等；没有部门、团队，那就大可先认为只有一个团队，日后再支撑多团队。

总之，先把核心业务做出来，判断产品优劣之后再说。

迭代内的用户故事排序

这个是具体开发级别的优先级排序方法，排序对象是敏捷开发的用户故事；排序人员是产品经理（PO）；排序依据是“此功能在此版本中完成的必要性”。

排序依据

值得注意的一点是：此功能很重要，不等于此功能必须在这个迭代中完成，它很可能只要在版本发布前完成开发就可以了（当然，越重要就越要在早一点的迭代中实现）。那么到底依据什么排序呢？

还是““此功能在此版本中完成的必要性””。这句话看起来很抽象，但是举一个例子就比较清楚了。

假设这个迭代做“用户，角色，权限”，那么应该先做哪些用户故事？注意这里要探讨具体的用户故事了，而非这三个史诗故事。

多半有这么几个用户故事：

增加用户，删除用户（不小心写错了用户名，而又不能修改），编辑用户（的邮件），批量增加用户，冻结用户（保留记录但禁止登录）……

增加角色，分配角色到用户，编辑角色（编辑角色的名字但不修改其权限），删除角色……

增加权限，分配权限到角色（透过角色进而分配到用户），分配权限到用户，删除权限……

可以看出有几个操作几乎是必需的：增加用户，批量增加用户，冻结用户，分配角色到用户，分配权限到角色。

而另外几个可能不太需要，比如：增加角色（可以先写死几种常见的角色），编辑角色（先写死日后再提供修改功能），增加权限……

还有几个几乎可以在早期版本中不要，比如：删除角色，删除权限（这个是基于火星人产品的实际情况分析的，读者的产品或许不同），虽然有了更好。

MoSCoW

有了大致的概念，MoSCoW是一种具体的操作方法来帮助我们实现。它是这几个词的缩写：

Must：这个迭代一定要做的。比如前面提到的“必需”的功能。

Should：应该做，但若没时间就算了。比如前面提到的“不太需要的”功能。

Could：不太需要的，但有了更好。比如前面提到的“几乎早期版本中不要”的功能。

这样，所有人就知道这个迭代中应该先做什么了。不过，这和大家真的会这样做还有点距离，经常需要一些技术手段，比如在故事板上，把TODO的任务按MoSCoW写在不同底色的故事卡片上，在真正完成M和S之前，不准开工W的任务。

值得注意的一点是，MoSCoW可以认为是一种思维方式，可以处理所有自相似的优先级排序问题，未必只是迭代内的（尽管它的设计初衷是）。

处理灰色地带

“如果我真的有时间，应该做那些Could的任务，还是干脆做下一个迭代的Must的任务？”这真是一个问题。

理论上说，下一个迭代Must的任务比这个迭代的Could任务更重要。但本人比较倾向于每个迭代集中注意力处理一类功能，而不要零星地掺杂日后的所谓重要的功能。这样可以有效地防止注意力分散导致的生产率下降。

如果你不太喜欢这个结论，其实更应该尝试一下流开发（日后会有详细描述）。流开发需要的管理模式对人的要求更高，但用好了可以应对很多这类问题。实际上火星人采用的就是流开发，而非迭代式开发。

不同优先级之间的继承关系及处理优先级变更

是否因为“用户”比“部门”优先级更高，而导致“增加用户”比“增加部门”更高，以及“删除用户”比“增加部门”也更高？（注意不是“删除部门”）

我们自己尝试的结果是：不一定。

尽管继承关系大致存在，但用户故事很难干净地从史诗故事继承优先级；而迭代也很难从版本直接继承。

所以要把精力放在实际业务上，而不要尝试找到万能方法。

另外，我们经常在几个月后发现原来一个本来认为不太重要的功能，还是必要的。为了处理这种不断的问题，我们会定期安排一个迭代，称之为“优化……”或“重构……”迭代，其中包含了处理遗漏的功能，还包括性能优化等其他事务。

有了这个迭代，就不是太担心偶然出现的优先级错误。另外也不必太担心这个迭代中处理的事情太杂乱而无从下手：是的，前面说过尽量只做几个有限的史诗故事来保证注意力集中，那是因为刚开始做版本的时候，大家不可能对所有故事都熟悉，同开始太多的事情会让大家更加混乱；但在版本的末期，所有事情都相对熟悉了，这种内容发散的迭代影响不大，反而能帮助大家重新整合和整体思考一下不同的功能。

关于产品路线图级别的排序在本文中有详细描述：http://blog.csdn.net/cheny\_com/article/details/7190301

关于MoSCoW：http://blog.csdn.net/cheny\_com/article/details/6358456

关于史诗故事及用户故事：这里边很多http://blog.csdn.net/column/details/agile.html，包括后来的三期线上沙龙的内容中经常提及。

敏捷开发一千零一问系列之三十五：如何获取准确需求？（兼谈精益创业）

目录(?)[+]

本文是敏捷开发一千零一问的第三十五篇。（栏目总目录）

问题

相比项目开发，产品研发中获取准确需求难度更大。因为客户和用户比较模糊，需求经常不知道问谁，而另外一个问题则是不知道该怎么问。

本人有幸参加过由Lean Startup Machine主办微软协办的精益创业培训课程（Lean Startup），课程中一个议题就是如何获取准确需求。

分析

为大众软件/应用收集需求的时候，首先应该注意，不要问有导向性的问题，比如你正在做一个电影订票应用，那么就不要这么问：

“你平时喜欢去电影院看电影吗？”

很多很少看电影的人也会回答“喜欢啊”。本人一年只去电影院看不到6次电影，但绝对会把自己归于喜欢看电影的人之列。

“你认为自己和朋友们会使用这种订票应用吗？”

“当然会考虑了”面对唾沫星子横飞表情激动的创业者，没有人会当头泼上一盆冷水的。

“如果每场电影只额外花1块钱，你会考虑付费购买吗？”

“当然了，如果软件很实用，我会考虑购买的。”谁会以铁公鸡的姿态说：“我才不会花钱呢”

那怎么办呢？

答案

正确答案如果用简单的话总结一下居然就几个字：用历史映射未来。

如何理解这句话呢？

就是让被提问者无需思考而只需回忆地说出以往发生的客观情况，以此来判断未来是否会发生类似情况。

比如上面的三个问题就可以换成：

“你平均一年会去电影院看几次电影？”

“平均两个月看一次吧，去得不多”

“你以前用过其他应用来购票吗？”

“XX算不算？用它订过几次～～的票；还有…………”

“你和你的朋友曾经为此付费吗？”

“我本人很少，就有一次……的演唱会，还真花过2块钱代购。不过我有个朋友，……”

用这种方法就不会得到过于乐观的答案。而且还可能挖掘出更细化的需求，因为用户在描述过去发生的事情时，会说出很多实际的细节；而描述未来做什么时，则往往非常含糊。

其他

当然这只是课程的一个小部分。课上还有一个重要概念是MVP（最小的有价值的产品）也很重要，是关于如何找到产品的第一个可发布版本的，很像我在敏捷开发产品管理系列中提出的基于市场来找到落脚点的观点。

# 敏捷开发产品管理系列之九：划分产品子系统

# 子系统定义

其实子系统不是一个严格的定义，这里指任何产品（当然还有一个问题，什么是一个产品……）的第一级功能目录，也就是最大尺度上的产品分解方法。

由于业界一直缺少标准分解方法乃至一些简单规则，可能一百个产品有一百个分法。

在开发火星人的过程中，“我们”偶然发现了一种易于掌握的方法。“我们”加上引号，是因为实际上是我在培训课的学员看了我们实际的分解结果后发现的；被他一语道破之后，我本人也恍然大悟。这种分解方法可能大家平时也在用，但却没有太注意。

# 下面的内容来自火星人内部的帮助系统，数据是由沙盘项目自动生成的，略有修改。因为有CSS差异，所以有少许格式问题。

此外，这是一个完整的“四步走”的第一步，在发完四篇博客后，会给大家一个链接直接到网上看。

**高级教程：四步生成故事树 [展开]**[要求补充帮助]

此教程适合在火星人系统中直接创建新产品的高级用户，建议第一次使用帮助时跳过。

此演示数据是依靠当前企业中的第一个产品自动生成的。想查看火星人官方示例，请在火星人首页使用公开的演示帐号登录后访问此帮助页面。

#### 合理与不合理的划分的判断准则

由于用户故事树是对用户需求的条目化，因此火星人主张**按功能而非架构对其进行分割**。下面是一个电子商城合理与不合理分割的例子：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 商铺管理子系统 | 商品管理子系统 | 广告管理子系统 | 收发货子系统 | ... |
| 前端界面层 |  | | | | |
| 业务逻辑层 |  | | | | |
| 数据层 |  | | | | |

合理与不合理的区别在于：

1. 1. 客户/用户可以理解合理分割的子系统存在的价值。
2. 2. 合理的子系统能直观表明产品的主要功能；不合理的产品则全都是“三层结构”或“四层结构”，看看不出是什么应用。
3. 3. 当大规模增加功能时，合理的子系统也会越来越多，能直观地看到功能增长；不合理的则不会发生变化，只是内部越来越庞大。
4. 4. 每当增加“小功能”时，合理的子系统只影响一个子系统；而1小时就能完成的小改进也可能影响不合理的所有子系统。
5. ……

尽管可能还有其他的价值观准则，导致不同的划分方法，但火星人认为显然以上四条有其不可忽视的意义，除非有等同价值的收益，不应该随意打破。

#### 推荐的划分方法：以主要用户及其业务关系的差异进行划分

在不同子系统中，主要用户不完全相同（一般都有重叠），**但主要用户间的业务关系相差很大**。比如上例中：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **子系统** | 商铺管理子系统 | 商品管理子系统 | 广告管理子系统 | 收发货子系统 | …… |
| 主要用户 | 店主，电商营运者 | 店主，（到店）买家 | 店主，电商营运者，（所有）买家 | 店主，快递，（已购物）买家 | …… |
| 业务关系 | 商铺的建立与运营 | 商品的分类，上架，展示等 | 商铺和商品的对外宣传 | 商品的交货 | …… |

下面是火星人自己的子系统：   注意！ 火星人是一个敏捷开发管理系统，下面列出的“产品管理”“敏捷开发”“测试”等是火星人自身的功能，而非指开发中的活动

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **子系统** | 产品管理 | 敏捷开发 | 测试 | 企业 | …… |
| **主要用户** | 产品经理 | 产品经理（计划，验收），项目经理和开发人员（故事板） | 测试经理/测试人员，产品经理 | 系统管理员 | …… |
| **业务关系** | 产品管理，需求管理，Wiki等 | 计划，开发（看板），评审 | 测试用例，自动化测试，测试结果 | 维护产品，团队，用户等 | …… |

#### 更大尺度上需求的划分：产品

若子系统的数量超过5～8个的合理范围，应考虑**在更大尺度上划分为产品**。

比如“火星人”处理单个企业内部不同工种的研发管理，而“火星云”处理在站点上，火星人运营者与多家企业、海量用户之间的互动。

火星云的子系统如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **子系统** | 站点 | 用户反馈 | …… |
| **主要用户** | 站点管理员，企业管理员 | 火星人支持人员，用户（企业的管理员，产品经理，项目经理，开发人员，测试人员……） | …… |
| **业务关系** | 建立，统计企业 | 用户反馈（缺陷，建议，问题……）及其处理 | …… |

产品和子系统没有非常严格的界限，常常是逐渐发展变化的。最初火星云是火星人产品中的一个子系统，但当整个火星人逐渐变得庞大时，火星云也独立成为一个产品。

在使用这种方法后，很想找到一种新的名词来代替“子系统”，因为很多人提到子系统的时候很可能想的都是前台界面和后台逻辑的划分，而最大尺度上的非功能分割。

看了上面的描述，“用户群”本来是一个不错的替代名词，但若没有看过本篇文章，一定会觉得一头雾水。**欢迎大家出出主意。**