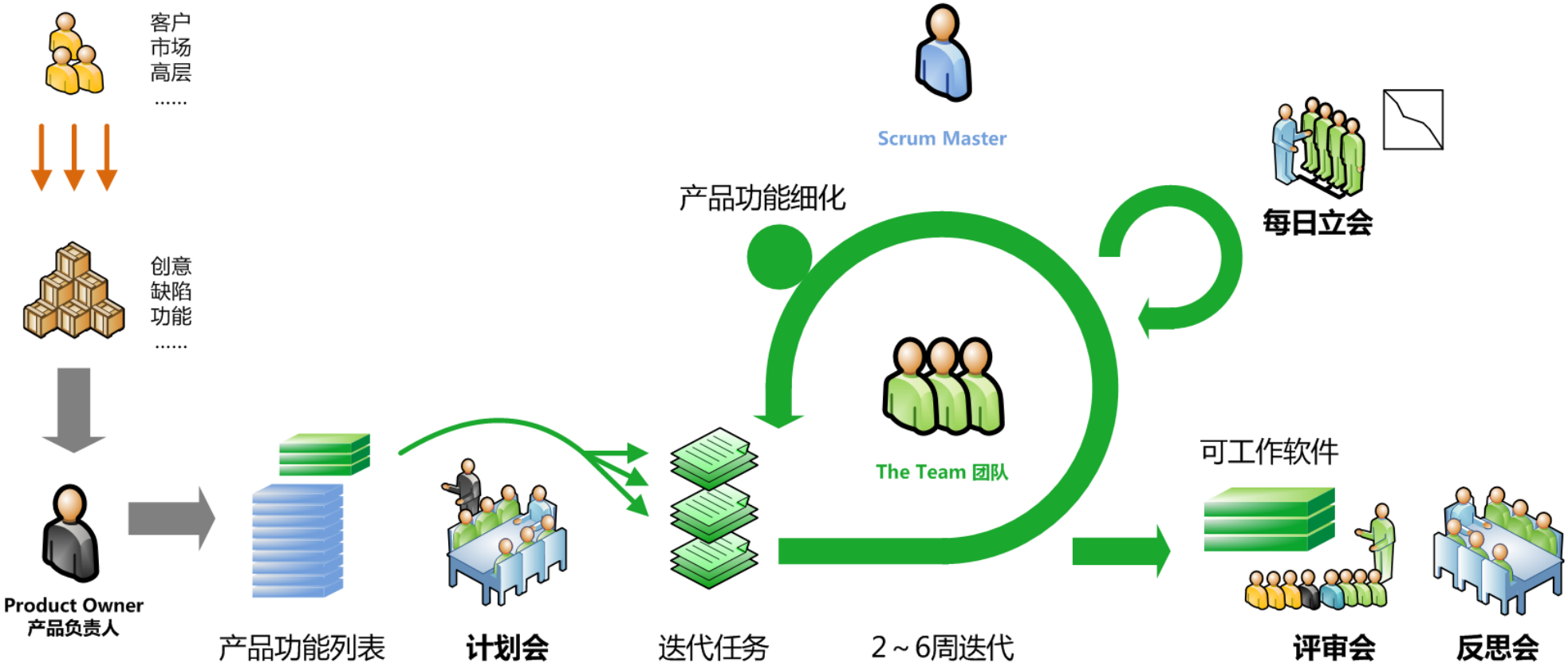
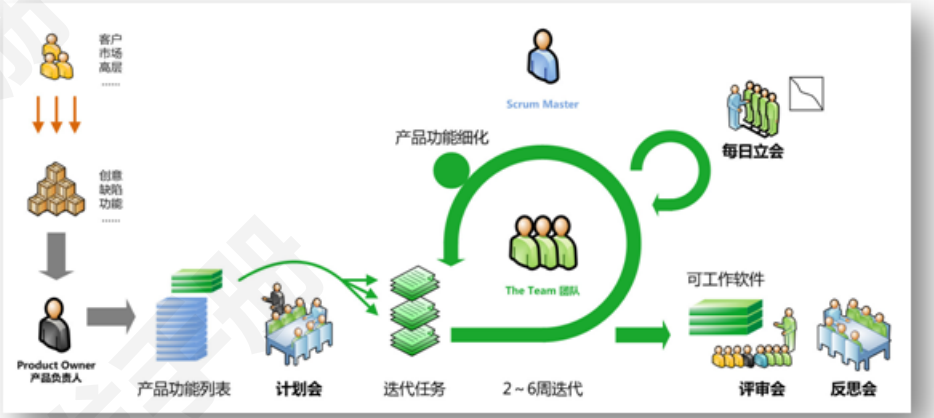
# 火星人敏捷开发手册

☺ 作为培训前的预习阅读。
☺ 打印并张贴在公司走廊上。
☺ 作为企业内部小组培训教材使用。



# 目录

# 火星人敏捷开发手册




## Scrum概览

Scrum是一种**兼顾计划性与灵活性**的敏捷开发过程，原词来自于橄榄球中的“带球过人”。在橄榄球比赛的每次冲刺前，都将有一个计划安排的过程，但冲刺开始后则由队员在原计划的基础上随机应变。

不同于瀑布模型将开发过程划分为需求、设计、编码、测试等阶段，Scrum将整个开发过程分为多次**迭代**（称为Sprint，冲刺），一般为期2～4周。

在日常工作时，**产品负责人**会维护一个**按优先级排序的“产品待开发项”**（Product Backlog），即从客户价值理解和描述的产品功能条目。

在每次迭代的第一天，召开**迭代计划会**(Sprint Planning Meeting)。产品负责人会逐一挑选最高优先级的部分进行讲解。团队可就需求细节、完成标准等进行询问，并逐条估算，放入本次迭代的开发任务中，直至任务量饱和。一旦迭代开始，这些任务将不会发生大的变化。

在迭代期内，团队将决定任务分配、所需的技术等，逐一完成任务。每天团队会进行一个简短的站立会议即**每日立会** Daily Stand-up Meeting，沟通当前进度、下一步任务和当前存在的问题，以借助团队的力量解决。团队还维护一张“**燃烧图**”（Burn Down Chart），即所有任务的累积剩余时间随开发进程与日递减的图形，以观察和预测所有任务是否会按期完成。

在每个迭代的最后一天，团队会召集**评审会**(Review Meeting)，邀请产品负责人等参加，对已经完成的产品功能条目进行评审，后者做出判断并给出改进反馈。当天还会召开**反思会**(Retrospective Meeting)，对本次迭代中的成功与失败之处做出总结，并在以后迭代中进行改进。

回目录

# Scrum是什么意思？

Scrum本意是指橄榄球中的“带球过人”

## 带球过人需要计划！

☺在球场上：在比赛每段的开始，双方都要摆开阵势，并计划本段的进攻/防守路线和策略，教练和队长都可以参与计划。

☺在软件开发公司：在每个迭代的开始，团队领导者都应该做好本迭代的计划，尤其是需求条目的优先级排序、选择本迭代的工作、设定必须完成的内容等。

## 带球过人需要灵活应变！

☺在球场上：当哨声响起，尽管队员们努力按照既定计划推进，然而场上瞬息万变，队员不可能实时按照教练或队长的指令亦步亦趋地行事，而是靠平时训练中形成的素养见机行事，达成目标。

☺在软件开发公司：在每个迭代开始后，团队领导不可能也不需要事必亲恭地者介入每件事情，而是应该由具体执行的人选择如何去做。团队领导要做好的是协调资源、解决困难、提供指导，以达成目标。

**Scrum中既有计划会、每日立会、评审会等计划和管理活动，又有迭代期内的灵活应变活动，是一种轻重结合的敏捷过程。**

回目录

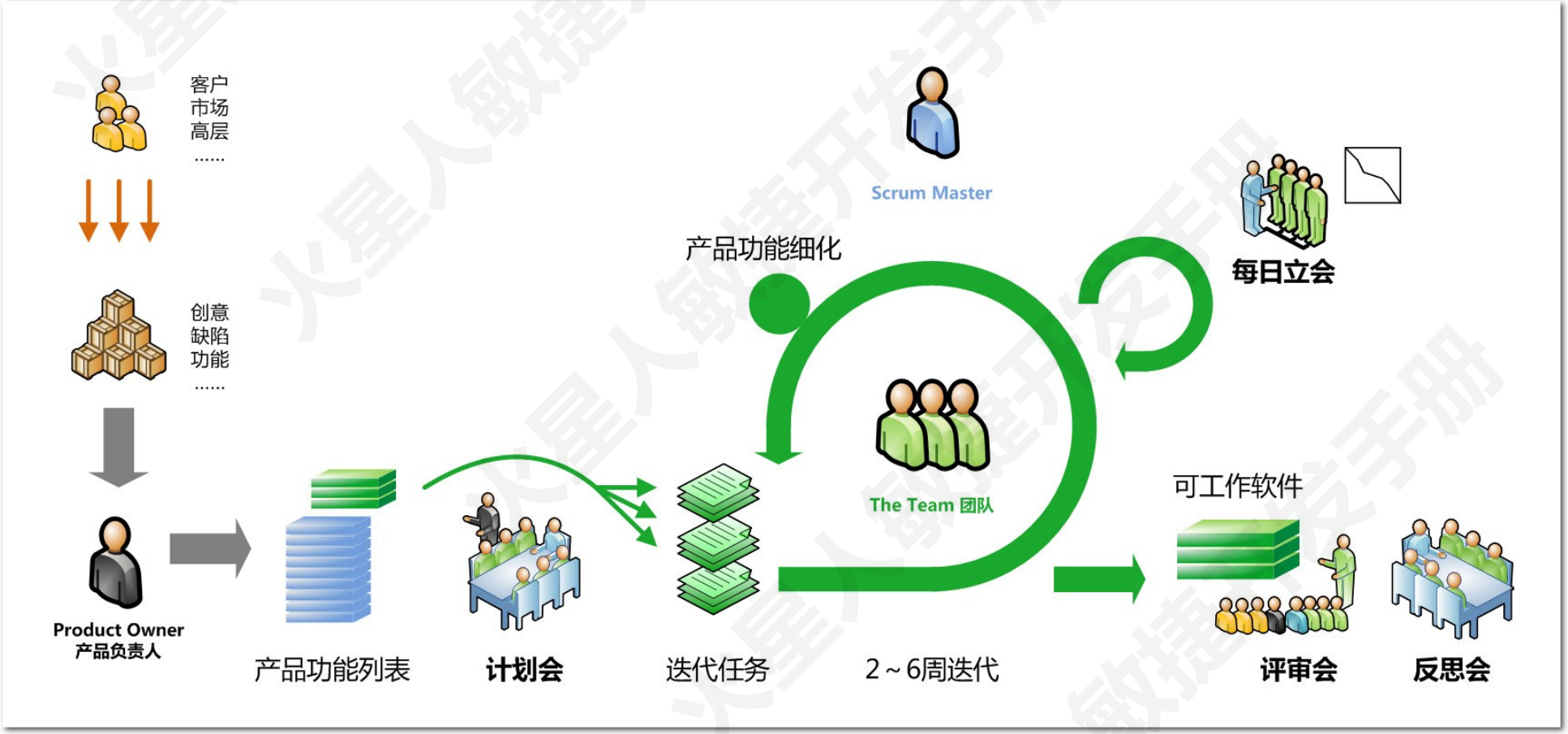
# Scrum敏捷方法一分钟扫盲

产品负责人**建立条目化的产品待开发项**，并进行优先级排序。

在**迭代计划会**上，产品负责人讲解本迭代要开发的条目，团队进行估算并放入下一个迭代。

团队在迭代内完成所列需求，每天都开**每日”立“会**以沟通进度和问题。

在迭代终点的**迭代评审会**上，团队向产品负责人等展示开发成果。




回目录

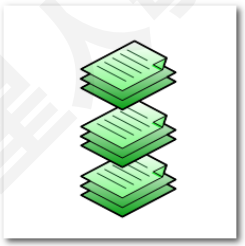
# Scrum敏捷方法中的工作产品

**产品待开发项 Product Backlog是从客户价值角度理解的产品功能列表。**

- ☺ 功能、缺陷、增强等都可以是待开发项。
- ☺ 一般以条目化的方式描述。
- ☺ 客户和用户必须能够理解。
- ☺ 描述怎样使用而非怎样制造。
- ☺ 整体上从客户价值优先级排序。
- ☺ 总工作量一般需要0.5～10人天。
- ☺ 高优先级的条目应有较详尽的描述，低优先级的条目可只有一个名称。

**冲刺待开发项 Sprint Backlog是从开发技术角度理解的迭代开发任务。**

- ☺ 在简单的纯软件环境中，可以直接把产品待开发项当作冲刺待开发项分配到迭代中。
- ☺ 在复杂的开发环境中，可以把一个产品待开发项分解为Web/后台……软件/硬件……程序/美术……等开发任务。



**可工作软件 Working Software是可交付的软件产品。**

- ☺ “可交付”在不同场景下差异很大，应视不同情况提前设定和选定交付标准。比如是否需要测试，是否需要性能优化，是否需要操作手册等等。
- ☺ 在正式产品中可能包括使用文档，甚至是纸质的。在新产品开发的初期，则可能只需要交付勉强可看到效果的产品。
- ☺ 产品负责人、用户代表等负责评审可工作软件。
- ☺ 若一个产品功能只是“差不多完成了”，会被视为不可交付。

回目录

# Scrum敏捷方法中的角色

Product Owner
产品负责人

**Product Owner（产品负责人）**负责产品需求的提炼、条目化、优先级排序。

**现实世界的产品负责人**

☺**部门经理、产品经理、策划人员**等都可能做产品负责人。

☺产品负责人是产品的指路人，必须对产品有长远的规划和深入了解，因此**不能简单地选择销售人员甚至客户作为产品负责人。**

☺大型产品如嵌入式产品和网络游戏，常常使用**有层级的产品负责人团队**，来解决广度与深度的矛盾，如产品总监-产品经理 / 主策划-策划团队。

回目录

Scrum Master

**Scrum Master（Scrum“大师”）**负责维护Scrum方法的秩序，并协助解决非技术问题。

**现实世界的Scrum Master**

☺Scrum Master的工作方式是**靠领导力而非权力**工作，因此首先应**服务于团队。**

☺一种人选是原来的**项目经理转型**，保留原有的管理和技术职能，但弱化指派任务、下达时间点指令等内容，而增强其组织协调能力。

☺另一种人选是企业原有的**过程改进人员**，协助不太了解Scrum的项目经理按照Scrum的方法工作，可以每人负责多个项目，接近全职的Scrum Master。

The Team 团队

**Team（团队）**以“自组织”的相对扁平方式进行管理，负责完成开发工作。

**现实世界的开发团队**

☺**实际团队常常不是“扁平的”**，而是仍有项目经理、小组长等职位。

☺工作中他们以“共同估算”“跨职能工作”“共同跟进”等方式**自组织工作**，而不是完全依赖层层指令。

☺项目经理、小组长的**领导、指导、协同职能大于其指令职能。**

#  与  的故事

敏捷角色背后的哲学

猪与鸡走在街上，鸡对猪说：咱们合伙开一家鸡蛋火腿三明治店如何？

猪想了想说：你当我是猪啊， 我要全身心投入，你却只是偶然参与。

**在敏捷开发中，不同角色各自对自己的工作内容拥有决策权，对于别人负责的事情，则只起到辅助、建议、挑战等作用。**

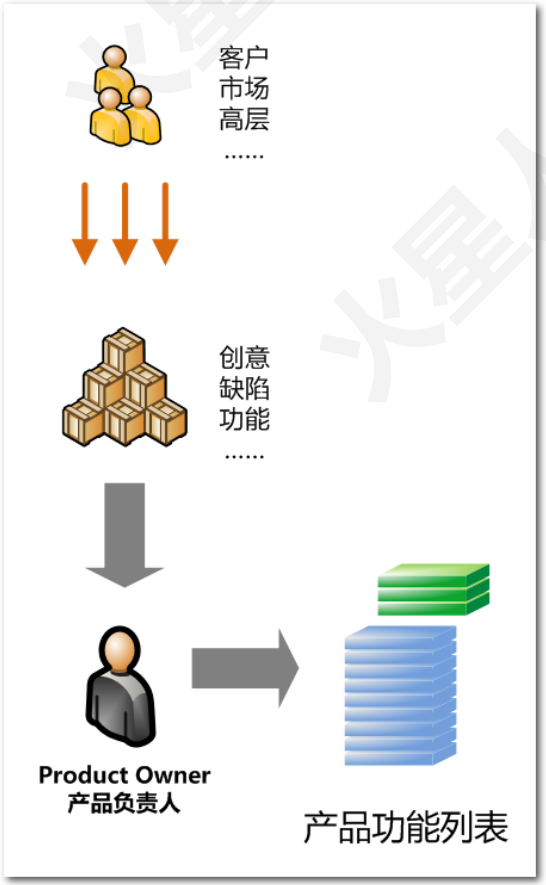

回目录

# Scrum过程-创建和维护产品待开发项（Product Backlog）

**产品功能列表（Product Backlog）**是一组条目化需求。

产品功能列表必须从**客户价值**角度描述，并**按优先级排序**。

典型的描述方法，就是极限编程中提到的**用户故事（User Story）**。

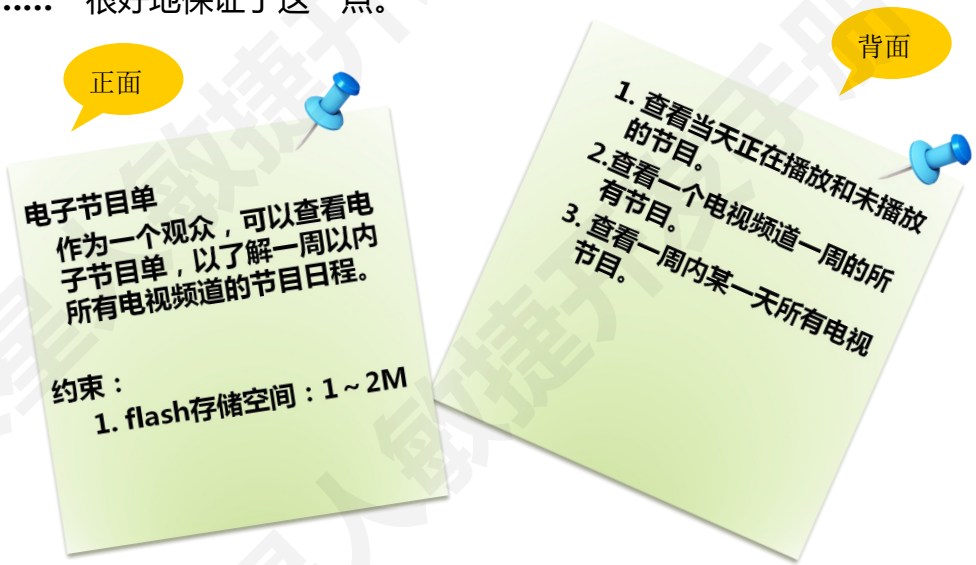



**如何编写产品功能列表**

☺**产品负责人**创建和维护产品功能列表。

☺需求必须进行**条目化管理**，才能进行分配、开发、跟踪，才能描述什么做完了什么没做完。

☺“客户价值角度”就是**描述用户如何使用，而不是描述技术层面如何实现。**比如“实现手写输入”“实现游戏排行榜”，而不是“编写数据库底层”。用户故事的语法**“作为一个……，可以……，以（以便）……”**很好地保证了这一点。




回目录

# Scrum过程-迭代计划会（Sprint Planning Meeting）- I

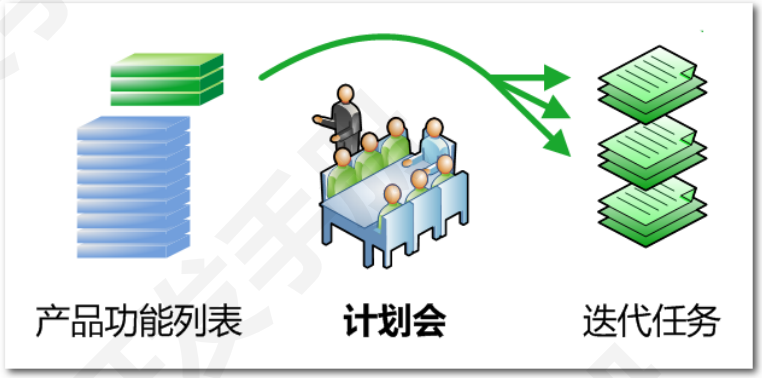
**迭代计划会**在每个迭代第一天召开，目的是**选择和估算本次迭代的工作项。**

产品负责人**逐条讲解最重要的产品功能。**

开发团队**共同估算故事所需工作量**，直到本迭代工作量达到饱和。

产品负责人**参与讨论并回答与需求相关的问题**，但不干扰估算结果。

## 产品负责人准备什么？讲解什么？




☺**会前准备：条目化的需求（用户故事），优先级排序，最近1～2个迭代最希望看到的功能。**会前准备至关重要，可帮助产品负责人理清头绪，不至于在迭代期内频繁提出变更、增加或删除故事。

☺**会上讲解：较难以文字表述的内容**，如游戏的文化背景，嵌入式设备的手感，OA系统背后的人事关系……讲解过程中团队可以随时发问，产品负责人要予以解答。若产品负责人感觉答案没想清楚，可推迟故事的开发，或将故事分解为“已想清楚的”和“未想清楚的”，后者推迟到下一迭代或更晚。

☺**两者关系：准备活动类似电影剧本编写**，它导致了这是不是一个好故事的基本问题；**会上讲解类似导演说戏**，它导致了这个故事我们能不能演好的问题。很多团队错误地认为已经有文档可读了，开会讲解太浪费时间，其实两者缺一不可。

☺一般一个团队只有70%的工作可以进行正式工作，因此**每个人月安排15人天左右即为饱和**，新团队则可能低至10人天。

回目录

# Scrum过程-迭代计划会（Sprint Planning Meeting）- II

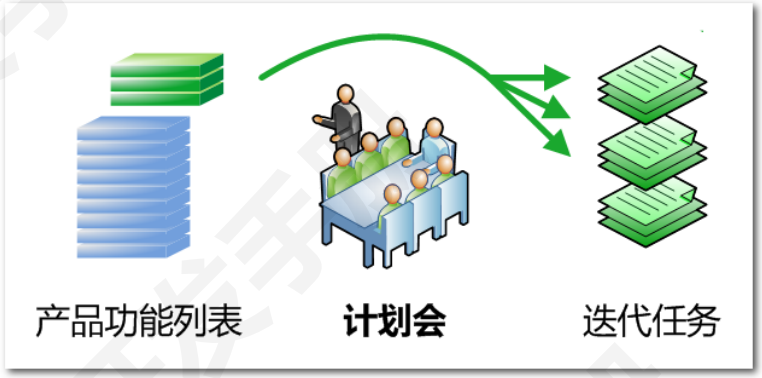
**迭代计划会**在每个迭代第一天召开，目的是**选择和估算本次迭代的工作项。**

产品负责人**逐条讲解最重要的产品功能。**

开发团队**共同估算故事所需工作量**，直到本迭代工作量达到饱和。

产品负责人**参与讨论并回答与需求相关的问题**，但不干扰估算结果。

## 团队怎样估算？




☺**共同估算：**在估算前不应该指定谁将开发这个故事，而是以接收故事的小组为单位集体估算，每个人提出自己的看法，大家综合。这样才能**以集体智慧完成任务。**

☺在估算全程，**产品负责人不能离开**，因为很多估算差异问题来自于“做什么，做到什么地步”，产品负责人需要予以解答，而不是让团队按自己的理解去做。

☺共同估算的目的不是一个数字上的统一，而是**用集体智慧和知识对“做什么，怎么做”达成共识。**

☺**共同估算是共同跟进的基础**，若不能共同估算，则后面的“每日立会”几乎不可能正常进行，因为大家只会关心自己曾经一起分析、思考、提问、设计乃至争论过的任务进展的怎样了，是不是和自己想象的一样。

☺共同估算的最佳方法是“**扑克牌估算**”，这看似很像一个小游戏，但却是Delphi估算的一个快速方法，同时实现了**匿名性与高效性。**

回目录

# 扑克牌估算（Planning Poker）

**估算扑克是几个潜在的任务承担者（如某个功能小组）共同估算的方法，他们一起听产品负责人讲解，一起估算，以达到利用集体智慧解决问题的目的。**

①每人各自估算后独立出暗牌，听口令一起开牌。

②数值最大者与最小者PK，其他人旁听也可参与。

③讨论结束后重新出牌和开牌。

④重复上述过程，直到结果比较接近。

回目录

**常见问题（在一个项目经理/小组长带新人的师徒制度中）**

**1. 为什么任务要分给组而不是个人？**

不分个人就打牌使得每个人都不得不思考，因为怕出错了牌又说不出所以然。这样即使日后他不做这个功能，也对这个功能很了解。

**2. 为什么不让最后领任务的人自己估算？**

因为他很可能因为不知道某代码可用、不知道某软件不行、不懂template（我们因此扔过1个人月代码）……而选择了错误的实现方法。

**3. 为什么不让师傅估算大家采纳，他不是最厉害吗？**

师傅的想法常常是徒弟们理解不了的——比如为什么不留在女儿国而偏偏去西天取经之类的——共同估算就是让大家在思考中对照自己的实现方法和师傅差异的过程。

**4. PO怎么还参加？不是不让别人干预吗？**

很多问题比如在游戏中“显示武林排行榜”，具体工作量可能不在于怎么做而在于做什么：凭什么排名？排多少名？实时排名还是每周排名？怎么显示排名？……因此PO不能写出一堆文档，然后以不便干预估算为名不参加敏捷计划会议。

PO可以挑战估算，比如：“这真的要这么久吗？我记得上次……”但要有理有据。其实实践中更容易看到的是，团队往往过于激进乐观，PO反而要让他们意识到完整的需求和要求，做出更现实的估算。估算不准确，PO也有责。

更多问题，请访问敏捷开发“松结对编程”实践之三：共同估算篇。

# Scrum过程-办公环境

开放办公环境是敏捷开发的一个符号。

与传统的开发中“背对背”地躲在格子间里边不同的是，敏捷开发提倡沟通与互动，除了每天的站立会议之外，随时随时发生的沟通也不可或缺。

而在旁边放置一个随时可以涂鸦的白板，则是团队的最爱。

回目录

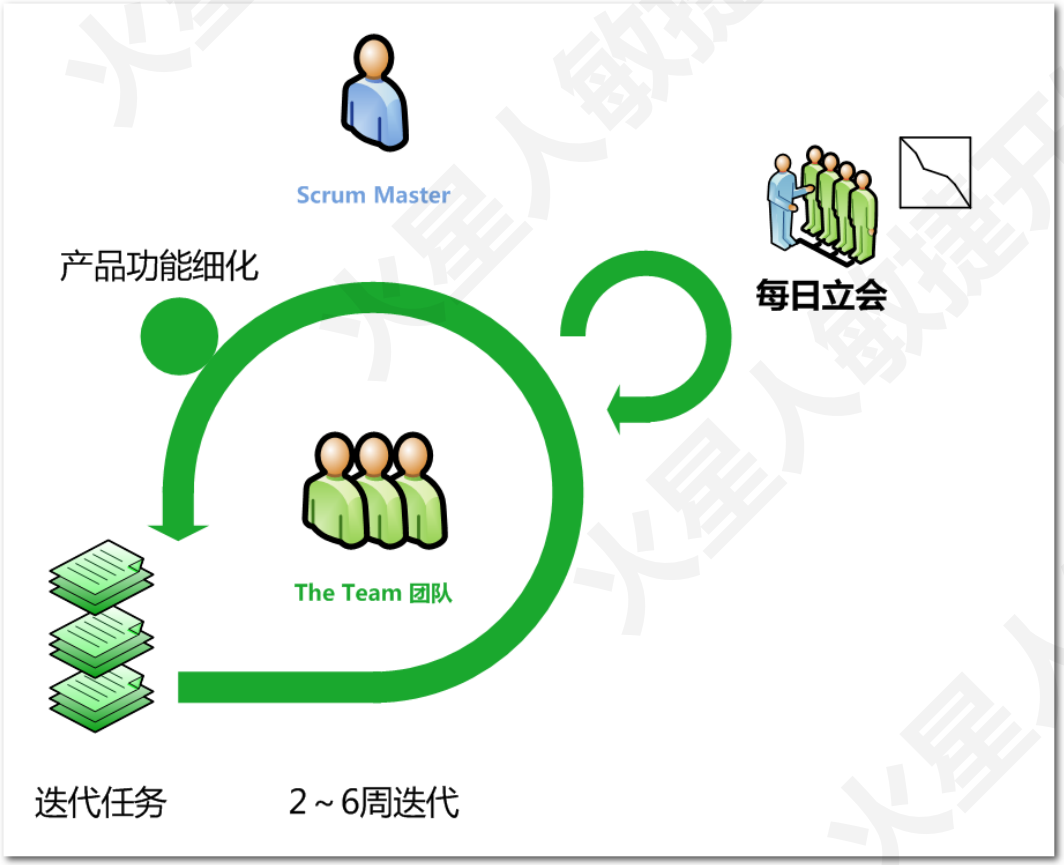
# Scrum过程-每日立会（Standup Meeting）

**队员认领任务**（或由组长协商分发），独立或与别人一起完成任务。

团队内部利用**每日立会**来沟通进度。

开发团队利用**燃烧图**来展示整体进度。

如无特殊原因，**迭代期内无变更**。




**什么是每日“立”会？**

☺由于每次会议只**持续10～15分钟**，人们习惯在工位附近的空地上站着开会，所以被叫做“每日立会”。

☺**每日立会上**每个人汇报三个问题：**我昨天做了什么，我今天要做什么，我遇到了什么困难。**由于小组曾经共同估算任务，所以如果出现偏差，可以沟通出现的问题以相互帮助。

☺**燃烧图（Burn Down Chart）**的横坐标是时间，纵坐标是本次迭代所有开发项的剩余时间，若时间到达终点而剩余时间也趋于零，则迭代顺利完成。

☺如无特殊原因，迭代期内产品负责人不会增加、改变、删除工作项，但却会协助开发人员进行**产品功能细化。**

回目录

# 拥抱变化？还是迭代期内无变更？

**正方-产品负责人**

☺敏捷提倡拥抱变化，当然要变更。

☺敏捷提倡不提前写太详细的需求和计划，当然要在开发中随需而变。

☺敏捷提倡“现场客户”，我总不能在现场把计划会上的话重新说一遍。

☺变更是客户提出来的，反映了客户价值，当然要变更。

**反方-团队**

☺敏捷提倡承诺，如果每次都变更，我们还怎么承诺？

☺Scrum要求“迭代期内无变更”，这还有什么可争的？

☺如果再变，我们下次只好提前预留时间！（嘘，他要不让预留，我们每个故事估算的时候都+1）

☺现在PO在计划会前准备就不充分，如果在允许变，他更不准备了。

**裁判-Scrum Master：一正一反，看得我头晕.....**

答案请在百度搜索：“迭代期内无变更”，前两条就是

回目录

# Scrum过程-评审会（Review Meeting）

小组向产品负责人**展示迭代工作结果。**

产品负责人**给出评价和反馈。**

以**用户故事是否能成功交付**来评价任务完成情况。

**怎样开展评审会？**


☺敏捷开发采用**时间盒（Time Boxing）**的方法，即限定时间而不限定范围。所以迭代不会延期，因为在迭代终点将放弃未完成的故事。

☺评审的**标准是整个故事是否已经达到交付标准**，而不是从其中分解出来的任务完成了多少，因此若一个故事“差一点就完成了”也算没有完成。

☺常常发生很多故事都已经开始开发，但都差一点完成的现象。因此**应按迭代内的优先级逐条开发和交付故事**。一般总是优先开发MoSCoW方法中必须完成和应该完成的故事，再考虑可以完成的故事。

☺一般在迭代计划会上设定每个**故事的完成标准**，如是否需要测试，是否需要考虑性能，是否需要说明文档等等。这些标准一般由项目组提前列好，每个故事只需要选中是否需要即可。

☺尽管有正式的评审会，但很多团队习惯在单个故事完成时，就让产品负责人进行**单个故事评审**，以确保交付时不会有“惊喜”。

☺评审会上发现的**问题或改进将被累积到产品待开发项**，也不会马上或在下一个迭代中开发，而是由优先级排序决定何时开发。

回目录

# Scrum过程-反思会（Retrospective Meeting）

| | | |
|---|---|---|
| 在**每个迭代后召开**简短的反思会。 | 总结**哪些事情做的好，哪些事情做的不好。** | **制定改进计划。** |

**怎样开展反思会？**

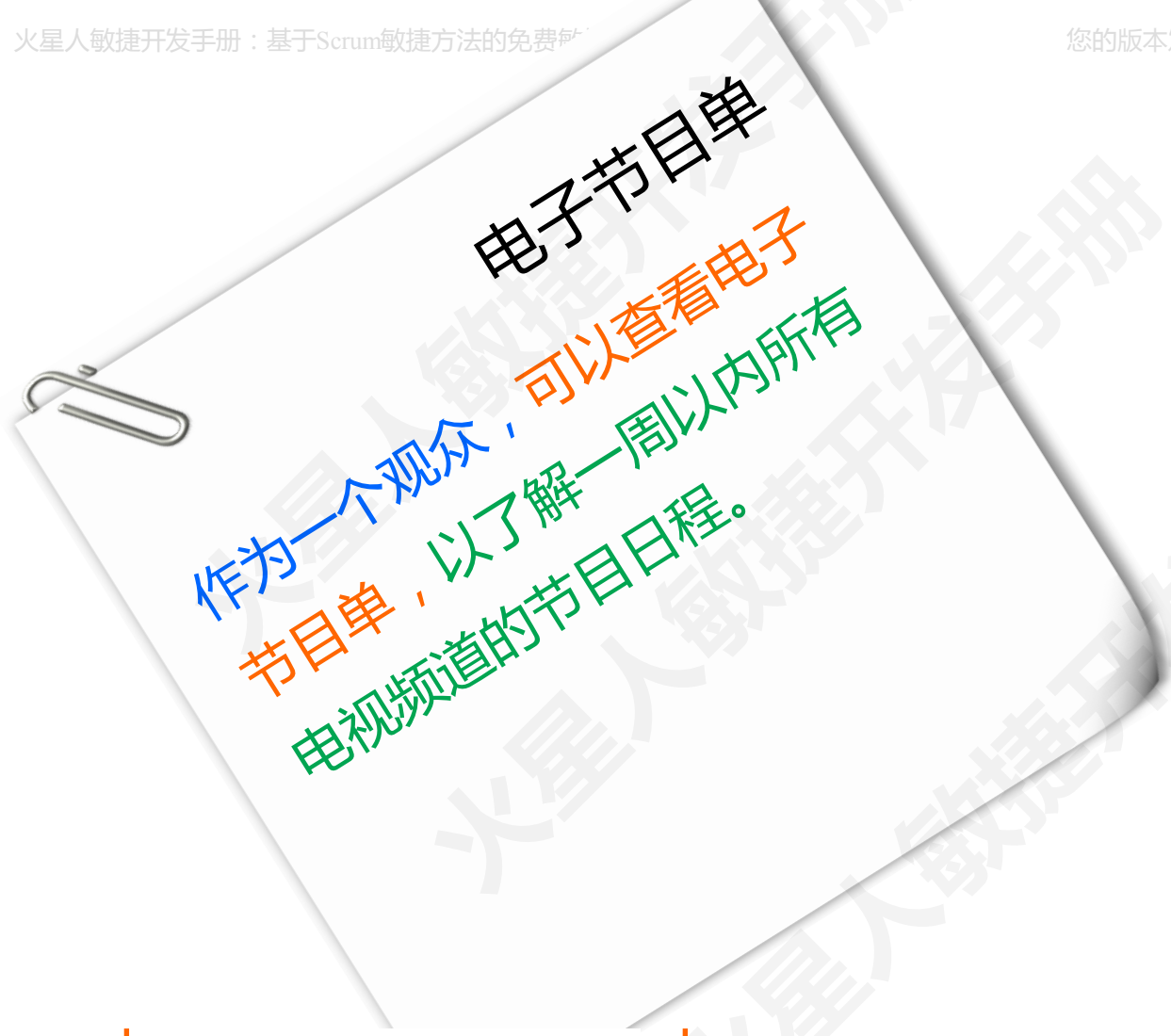

☺反思会是Scrum中最难以实施的活动之一。

☺反思会上讨论三个问题：**我们上个迭代有哪些事情做的好希望继续，那些事情做的不好希望改进，有何改进计划。**

☺经常出现一些问题多次被提到，但却始终没有解决。应该**每次仅就1～3个关键问题做出可行的解决方案**，在下一个迭代执行改进。“可行”的概念包括两个含义：第一是方法简单，影响面小，见效快；第二个是目标不要激进，而要现实可行，积少成多。

☺如果必要可以执行**领导回避制度**，即具有管理职能的人不参加反思会，即使这些人是产品负责人，项目经理，乃至Scrum Master。

回目录




# 用户故事 I：何为用户故

☺按“**作为一个……，可以……，以便……**”样式和思路写成的用户需求，就是用户故事。

☺这种样式是技法层面的东西，它保证了无需太多思考，用户故事中即可全面包含角色、功能、价值这三个要素。

☺要想写好用户故事，要改变那种面向功能而非客户需求的纯技术观念。

**角色切记不要总是写“作为一个用户”，而是要把用户区别对待。**
这样才能更好地理解他们使用什么功能，如何使用，为何使用。

**功能即用户能亲自执行的操作。**
应区分用户操作和产品功能之间的关系，因为产品功能可能也提供了用户所需的价值，但却极可能不便于操作。

**价值是完成操作后，客户所得到的好处。**
价值里边，常常要带有一点褒义词，或有一些吸引人的内容，比如“高效地……”“……可以节省话费”等。

更多分析，请见：敏捷开发用户故事之一：何为用户故事

回目录

# 用户故事 II：面向用户价值编写故事

哪些是好故事？为什么？

### ☺ 显示排名

☺作为一个排名靠前的付费玩家，可以通过显示排名，以便让自己在服务器中的地位获得认可（以刺激消费）。

### ☻ 显示排名

☻作为一个玩家，可以通过显示排名，以查看自己在服务器中的地位。

### ☻ 调动人员权限调整

☻作为管理员，可以在有人员调动时查询和修改其权限，以保证权限顺利转移。

### ☺ 权限查询

☺作为管理员，可以查询和调整所有用户的权限，以改变某些所需调整的用户的权限。

### ☺ 防打扰功能

☺作为一个用户，可以选择‘认可所有相似操作’，以便同意或禁止连续的修改硬盘、访问网络的操作。

### ☻ 防打扰功能

☻作为一个用户，可以在安装软件时选择‘认可本次安装操作’，以便一键完成正常的安装操作。

更多分析，请见：敏捷开发用户故事之二：面向用户价值编写故事

回目录

# 用户故事 III：用户建模

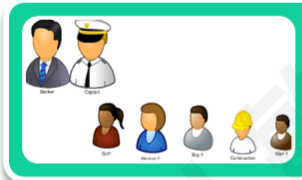
☺用户建模的目的，是理解**哪些用户**可能来使用产品或访问网站，他们为解决**哪些问题**而来，为达到**哪些目的**而来。

☺良好的用户建模，可以保证不同用户都能**方便地访问其功能**，使用产品后也更容易**获得其特定的价值**。

☺将大量功能无序地摆放到大量用户面前，不但不会增加价值，反而令用户无所适从。

**用户建模四部曲**


列出尽可能多的用户


识别关键用户（购买决策者/主要使用者）


合并次要用户


面向关键用户编写故事
关键用户的故事处于更高优先级

更多分析，请见：敏捷开发用户故事之三：用户建模

回目录


# 用户故事 V：用户故事的分类

☺用户故事一般被按尺度分为**史诗故事和普通用户故事**。若要更精细化地管理，则可能包含颗粒度和可见性等更多维度，并因此而产生出更多的分类。

☺这些分类本身没有固定的方法，也没有优劣之分，但都必须**服务于特定产品在需求管理上的某种目的。**

☺比如为了**在不同尺度上观察整个产品的功能，又想让合适的人看到合适的故事**，就可能产生下面的分类方法：

| 颗粒度 | | 客户可见 | 产品经理可见 | 开发团队可见 | |
|---|---|---|---|---|---|
| **产品功能描述** | **数据级别** | 史诗 | | 底层数据 | |
| | **操作级别** | 功能 | | 底层功能 | 重构 债务 |
| **版本发布描述** | **增强级别** | 增强 外部缺陷 | 内部缺陷 | 底层增强 | |

某项目的实际分类方法

更多分析，请见：敏捷开发用户故事系列之五：用户故事的分类

回目录

# 用户故事 VI：产生与组织结构


☺一种简单直观的**产生用户故事的方法**，是在项目开始的时候，列出用户要管理的数据及其操作。

- 所谓 **数据**，就是左图中“用户、角色、权限”这三种东西，就是要管理的**业务数据**，这里权且记下用户有“3个史诗故事”要实现。
- 所谓 **操作**，就是对“用户”这种数据，应该有增、删、改、查、加入角色……这些称之为**业务操作**，这里权且记下对用户，会有“5个用户故事”。

这些对应关系不是绝对的，但却可以在早期帮助迅速勾勒出产品轮廓。

这个史诗故事，是一个**业务数据**

这个用户故事，是一个**业务操作**

这个用户故事，是**两个业务操作**

客户理解这些数据或操作，并会在日常工作中管理这些数据，执行这些操作。

这是个对史诗故事的增强，**不是**任何业务操作 } 为研发人员做的增强，客户无法理解，也不会用到。

这是个对用户故事的增强，**不是**任何业务操作 } 为客户做的增强，但不是一个业务操作，只是改善了界面。

更多分析，请见：敏捷开发用户故事系列之六：用户故事的产生与组织

回目录


# 用户故事 VII：用户故事，MVC，FPA

☺**敏捷开发用户故事、Asp.net MVC、功能点分析法（FPA）**在一定程度上具有相同的目的：**作为用户需求与开发人员工作的桥梁**，只是侧重点有所不同。

- 敏捷开发用户故事关注**需求收集与管理；**
- Asp.net MVC关注**架构设计与实现；**
- 功能点分析法关注**估算。**

☻ 若割裂地管理它们，就可能需要为三种管理目的，分别维护三套管理数据，学习三种知识体系。

☺若能将它们联合应用，就可能用一种组织方式贯穿性地管理三者。

| | 用户故事 | MVC | FPA |
|---|---|---|---|
| **数据** | 史诗故事 | Controller | ILF内部逻辑文件 |
| | | | EIF外部逻辑文件 |
| **操作** | 普通故事（功能） | Action | EI外部输入 |
| | | | EO外部输出 |
| | | | EQ外部查询 |

更多分析，请见：敏捷开发用户故事系列之七：用户故事与MVC

回目录

# 敏捷计划总流程

**敏捷计划并非只局限于迭代计划会，而是会贯穿整个产品研发的周期中。**

下面是围绕计划会前后发生的具体操作步骤，某些标注为“会前”的活动未必是会前仓促完成，往往是在之前更早的时间点上早有准备。

## 会前

- ☺ **项目经理**排定每个迭代的工作日历，以便计算整个团队的**可用时间**。
- ☺ **产品经理**列出每个迭代的**周期、大致目标。**
- ☺ **产品经理**从当前的产品待开发项（Product Backlog）中选择一些故事组成**意向表**（Willing List），准备在会上讲解。

## 会中

```
//先估算后分配，以便所有人积极参与。
项目经理.核对(日历);
产品经理.概述(迭代计划);
产品经理.概述(上一迭代, 本次迭代, 今后迭代);
foreach (var 故事 in 意向表.OrderBy(重要性))
{
    产品经理.讲解(故事);
    团队.估算(ref 故事);
    本次迭代.Backlog.Add(故事);
    if (本次迭代.Backlog.总工作量 > 团队.可用时间)
        break;
}
return 本次迭代.Backlog;
```

## 会后

- ☺ 团队成员认领或项目经理协助，将用户故事分配到个人或小组。

回目录

# 敏捷计划I：可用时间计算

**一个人月到底能完成多少工作量**？“当然是22天了！”

那么，计划会、评审会的时间算不算？中途处理突发任务的时间算不算？写文档的时间算不算？做设计的时间算不算？帮助徒弟算不算？中午打瞌睡的时间算不算？.....以及一个经常问到的问题：估算的时候要留余量吗？

为了解决这些复杂的问题，前人已经找到了较为成熟的方法，**步骤如下**（参考页末图中数据）：

- ☺ 先减去Scrum会议的时间，一般是计划会-1天，评审会-1天
  图中2.1计划会和2.29评审会各-1扣除。
- ☺ 再减去确定无疑的出差、培训、请假.....的时间，加上确定无疑的加班时间
  图中cheny在6/7两天有一共-1.5天假期，yock在13/20～24有-6天假期；都没有加班 
- ☺ 剩下的时间，新团队×50%，老团队×70%，得到预计可用时间
  这样两个人最后剩下30.5天可用，如果×70%，结果是21.35天

估算故事时，**不再留任何余量**，按全时工作计算。所有需要在迭代交付时以PO指定的标准完成用户故事的事情都计算在内。

| 2012 年 2 月 | 总计（人天）<br>预计可用 21.35<br>（可用率 70%） | | | | 1<br>W | 2<br>T | 3<br>F | 4<br>S | 5<br>S | 6<br>M | 7<br>T | 8<br>W | 9<br>T | 10<br>F | 11<br>S | 12<br>S | 13<br>M | 14<br>T | 15<br>W | 16<br>T | 17<br>F | 18<br>S | 19<br>S | 20<br>M | 21<br>T | 22<br>W | 23<br>T | 24<br>F | 25<br>S | 26<br>S | 27<br>M | 28<br>T | 29<br>W |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 总计 | 30.5 | -11.5 | 0.0 | -11.5 | -2 | | | | | -0.5 | -1 | | | | | | -1 | | | | | | | -1 | -1 | -1 | -1 | -1 | | | | | -2 |
| cheny | 17.5 | -3.5 | 0.0 | -3.5 | -1 | 上班 | 上班 | 休息 | 休息 | -0.5 | -1 | 上班 | 上班 | 上班 | 休息 | 休息 | 上班 | 上班 | 上班 | 上班 | 上班 | 休息 | 休息 | 上班 | 上班 | 上班 | 上班 | 上班 | 休息 | 休息 | 上班 | 上班 | -1 |
| yock | 13 | -8 | 0.0 | -8.0 | -1 | 上班 | 上班 | 休息 | 休息 | 上班 | 上班 | 上班 | 上班 | 上班 | 休息 | 休息 | -1 | 上班 | 上班 | 上班 | 上班 | 休息 | 休息 | -1 | -1 | -1 | -1 | -1 | 休息 | 休息 | 上班 | 上班 | -1 |

关于“故事点是怎么回事”以及“21.35天到底能完成多少故事”的问题，请参考：敏捷开发绩效管理之五：敏捷开发生产率（上中下三篇）

回目录

# 敏捷计划II：迭代计划

Product Backlog里边有干不完的活……

每个迭代（Sprint）挑出最重要的部分……

一个迭代连着一个迭代……

仅仅这样工作，尽管团队现在总是有活干，但是产品未来究竟会如何，却是看不清楚的。因此需要一个**迭代计划来支撑对未来的预见。**

| ID | 标题 | 开始日期 YYYY/MM/DD | 结束日期 YYYY/MM/DD | 长度 自动 | 计划工作量（人天） 理想值[预计值] | 目标 |
|---|---|---|---|---|---|---|
| 351 | 计划会完善 | 2012/2/1 | 2012/2/29 | 29 | 30 30.5 [21.35] | 发布一个包含完整计划会过程、且包含适度的帮助与提示信息的版本。 |
| 347 | 迭代计划完善 | 2012/1/1 | 2012/1/31 | 31 | 29 23 [16.1] | 发布一个包含创新性视图的迭代计划、计划会界面的版本。 |
| 282 | 基本迭代，BDC | 2010/4/1 | 2010/4/30 | 30 | 30 44 [30.8] | 交付一个能讲故事分配到冲刺中/能生成BDC的版本。 |
| 281 | 基本故事显示 | 2010/3/2 | 2010/3/31 | 30 | 30 44 [30.8] | 交付一个能完成故事增删改查/故事讲解的版本。 |

**为每个迭代起一个简短的标题外加设定一个面向客户价值的目标是非常好的实践。**

如果标题和目标找不到合适的，表明大家其实并不知道自己在做什么，也就是众多的“当前最重要的故事”很难最终被汇聚为有意义的客户业务，进而提供客户价值。

深入讨论请参考：敏捷开发产品管理系列之一：序言及设立迭代目标

回目录

# 敏捷计划III：迭代意向表

**在迭代计划会前，产品负责人一般会从众多故事中选择其中一些形成一个迭代意向表（Sprint Willing List），作为计划会讲解的备选故事。**

- ☺ 迭代的意向表，是迭代目标在用户故事中的具体体现。
- ☺ 因此意向表一般并非“当前最重要的用户故事”组成的乌合之众，而往往是一些相关性强，完成后具备相对完整的业务功能的一组故事，称为“**故事群**”（一般比传统的史诗故事规模更大）。
- ☺ 意向表的故事往往有一个**预估值**，以便产品经理能够大致估算出下个迭代能完成哪些故事；预估值来自于一般仅由少数业务骨干参与的预估会议。
- ☺ 意向表中的故事、估算都不是最终结果，在计划会上讲解故事-估算故事活动后，项将有可能被更新。


关于“如何进行优先级排序”以及“故事群”，请参考：敏捷开发用户故事系列之四：优先级排序
关于“预估会议”，请参考：敏捷开发产品管理系列之五：预估会议

回目录

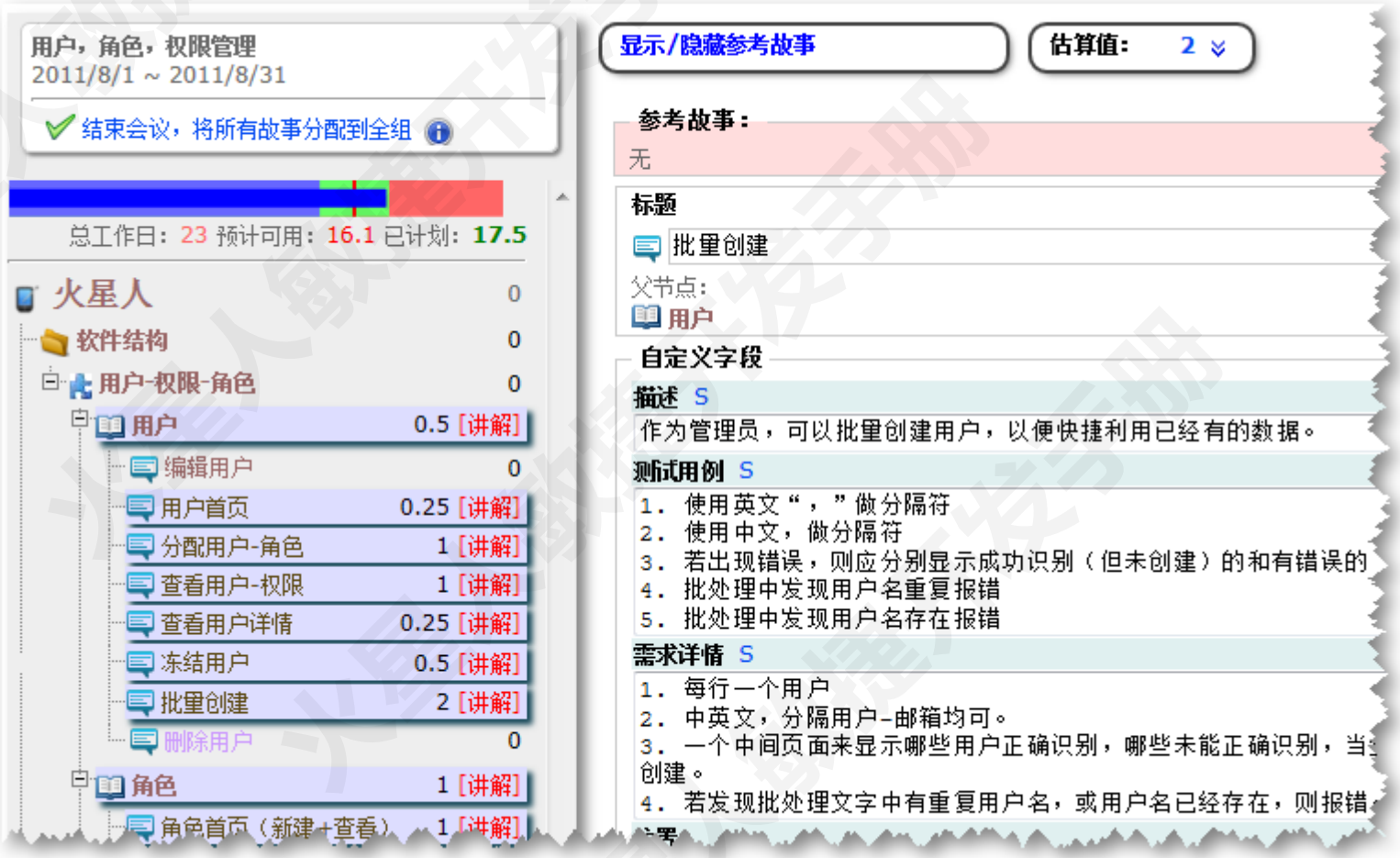
# 敏捷计划IV：故事讲解与估算

**在迭代计划会上，产品负责人逐个讲解用户故事，团队逐个进行估算。**

☺ 讲解顺序应先业务后技术，先总体后细节，先过去后未来，帮助团队系统地把握产品功能。

☺ 团队应简要记录需求详情，有些团队在问答环节甚至能当场形成基本的测试用例雏形。

☺ 在估算之前，不应该事先指定开发负责人，以便提升整个团队对估算的积极参与。




更多细节，请参考之前页面：

产品负责人讲解什么？

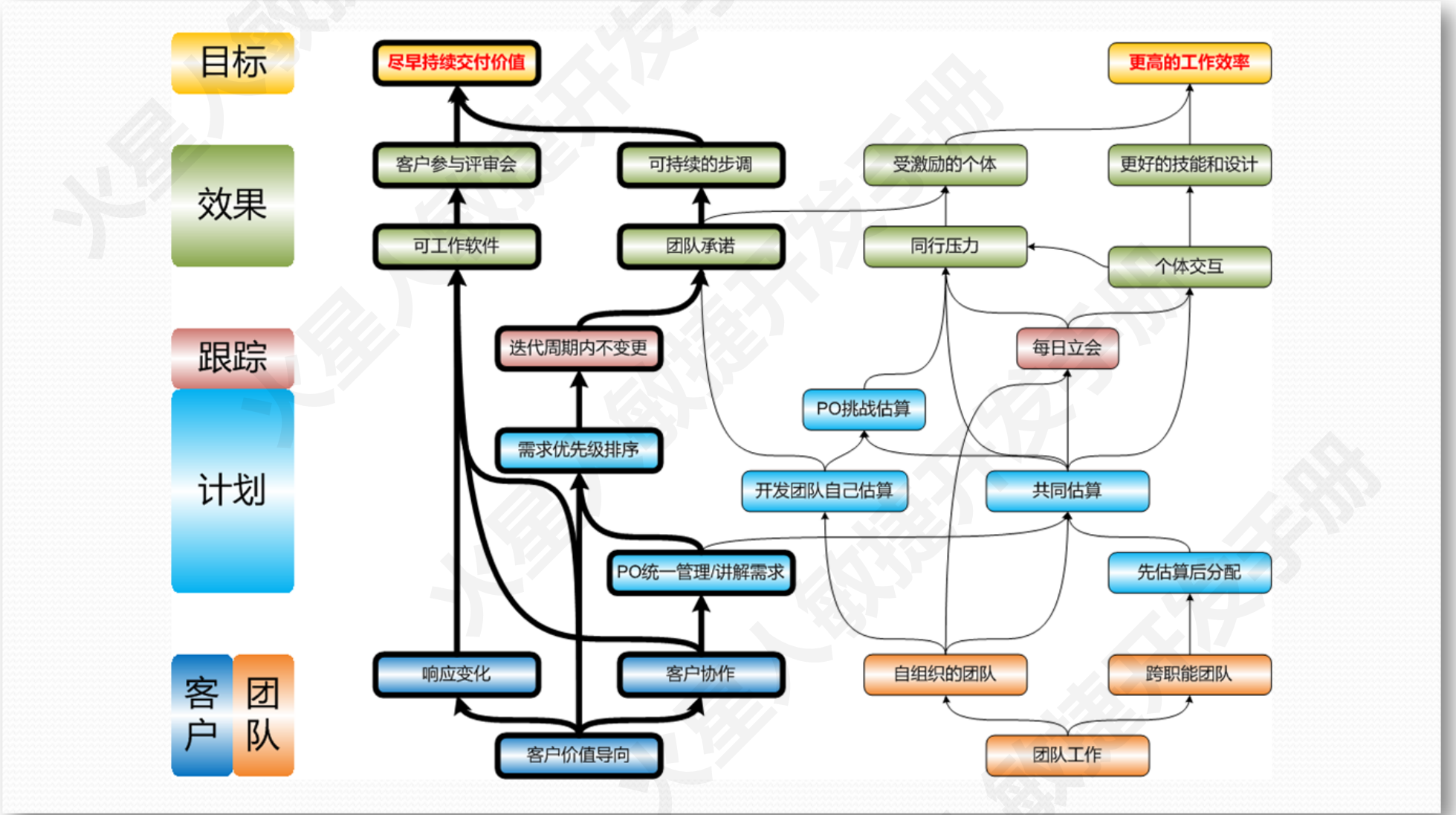
团队怎样估算？

扑克牌估算

回目录

# 敏捷生态系统-需求管理

敏捷生态系统表明了敏捷开发中各种实践的依存关系。当一个实践很难实施时，往往是另外一个实践未能很好实施造成的。




回目录

# 敏捷生态系统-需求管理

## 客户价值导向-可工作软件-响应变化

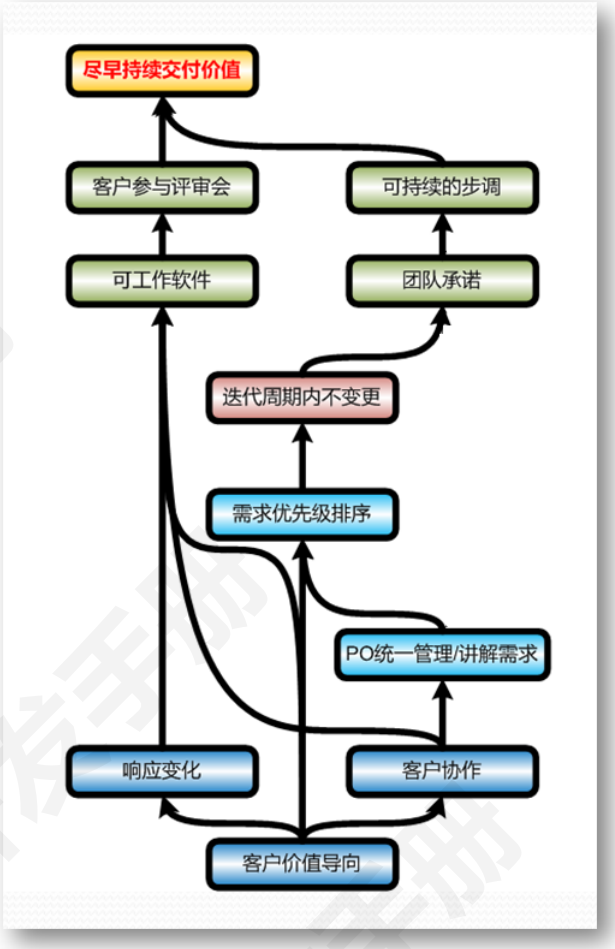
☺**客户价值导向**是敏捷开发需求管理的主要思想。

☺敏捷开发相信密切与**客户协作**比编写详尽的文档更能弄清楚客户的需求；而利用阶段性的**可工作软件**外加邀请**客户参与评审会**，比让客户评审需求文档更容易让客户正确地补充需求和验收产品。

☺由于相信变化后的软件一定比变化前的对客户更有价值，所以敏捷崇尚**响应变化**。提供可工作产品来引导客户变化，可保证客户更能正确翔实地描述变化；而迭代交付则使得重要的、必要的变化可以提前交付，而不是像瀑布模型一样最后才发生。

☺通过**需求优先级排序**和迭代交付，首先可保证重要的需求一定可以交付到客户手中；其次可以保证重要的变更来临时，可以放弃尚未开发的次要需求作为交换；再次可以保证产品负责人（PO）会优先分析重要的需求，不会让它们在模糊状态进入开发。

☺只有最高优先级的需求才会进入下一迭代，因此很少有变更比它们更重要，而且这些需求也被较深入地分析过，产品负责人就有信心承诺**迭代期内无变更**，以换取**团队承诺**，进而保持交付以**可持续的步调**发生。



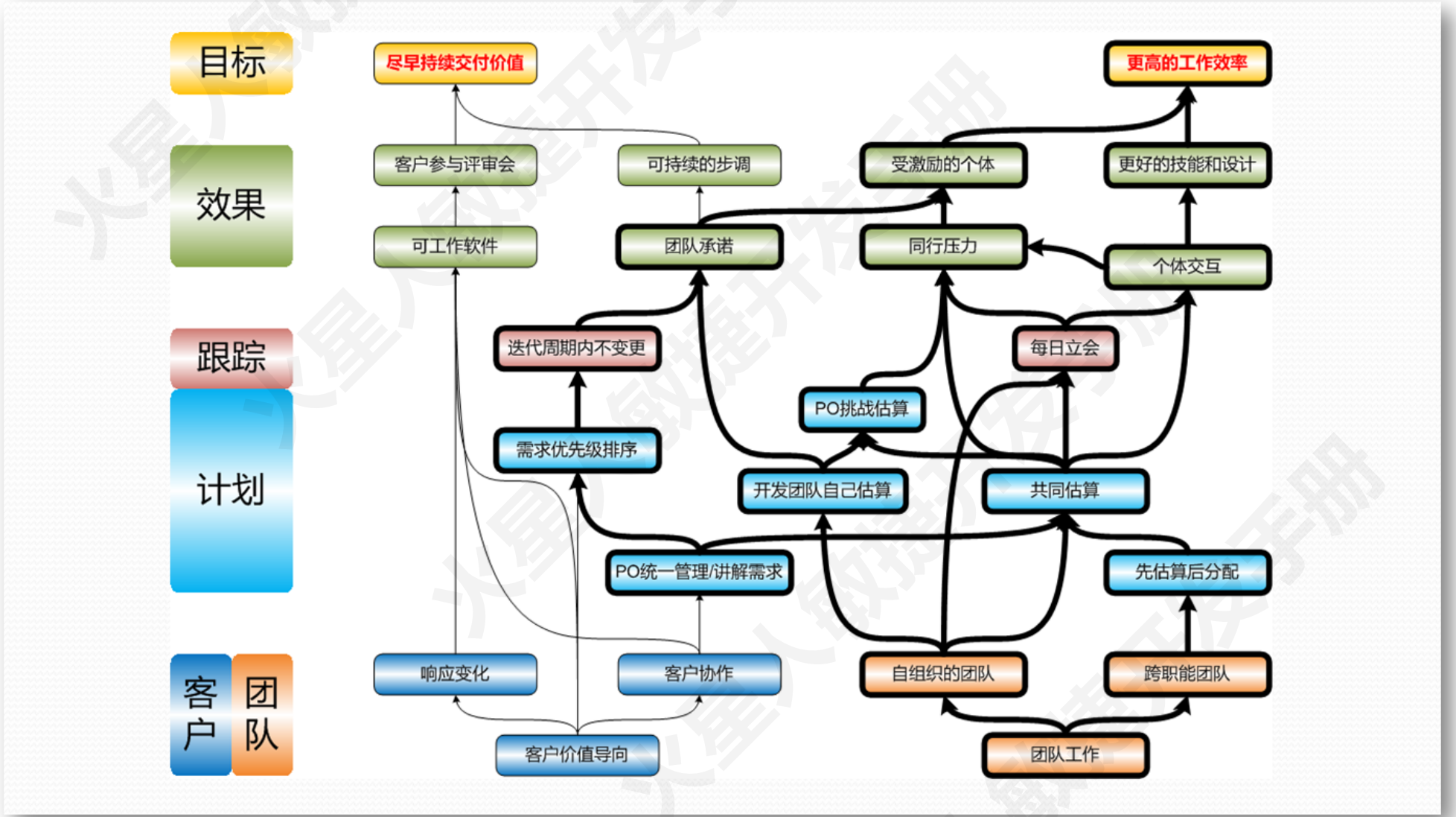

☺**拥抱变化**是一种由客户协作、优先级排序、可工作软件等各种实践支撑下的、主动的可控过程，而不是被动地“被变化拥抱”，“迭代期内无变更”和“拥抱变化”的对立统一，必须建立在这些实践的基础上。

更多分析，请见：敏捷开发生态系统系列之一：序言及需求管理生态（客户价值导向-可工作软件-响应变化）

回目录

# 敏捷生态系统-计划跟踪 I

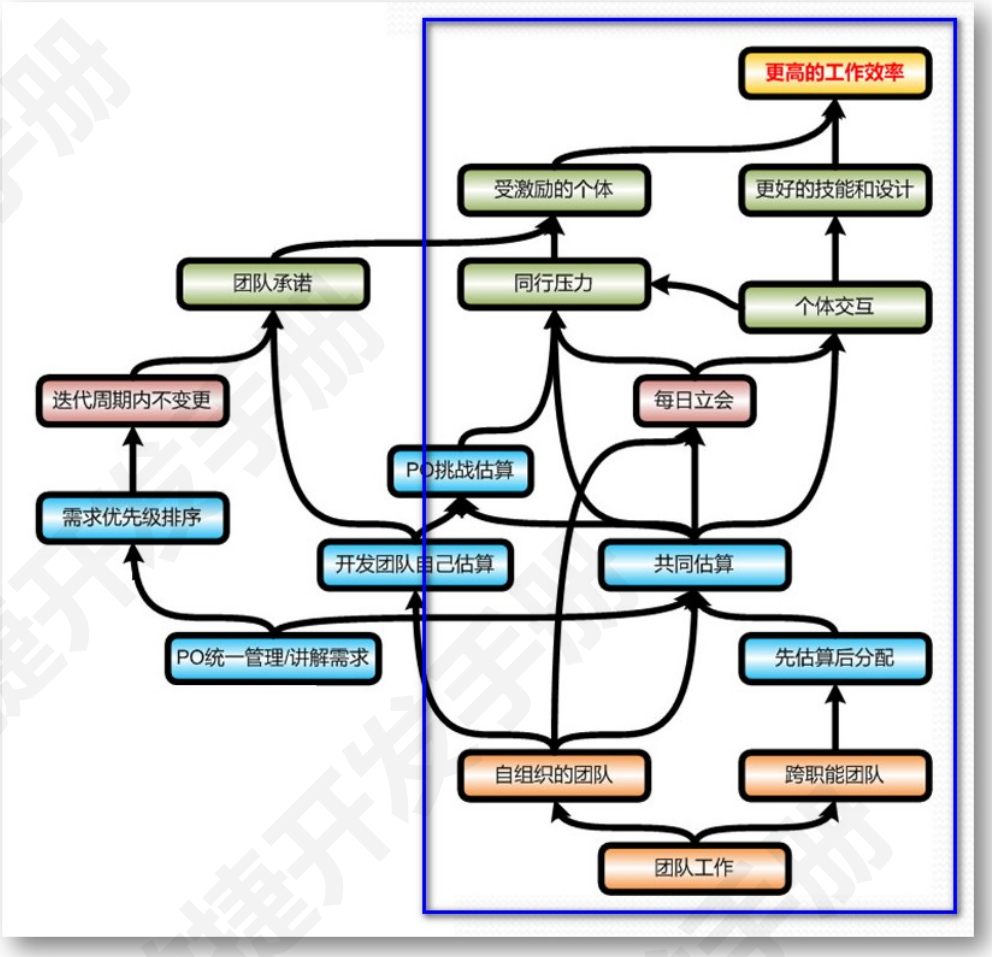
敏捷生态系统表明了敏捷开发中各种实践的依存关系。当一个实践很难实施时，往往是另外一个实践未能很好实施造成的。




回目录

# 敏捷生态系统-计划跟踪 II

## 跨职能团队-共同估算-每日立会-同行压力




☺**跨职能团队**的整体思路是“每个人可以做每个工作”。好处是消除了资源分配的瓶颈和造成队员无法互助的分工壁垒。

☺任务应该**先估算后分配**给个人，以便整个团队（或至少其中的某个小组）都对其保持兴趣，才可能进行真正的**共同估算**。

☺**共同估算**（一般采用扑克牌估算）中人们利用整个团队的智慧充分讨论和确认任务的实现目标和方法，因此**PO会统一管理/讲解需求**，以防止个体理解差异。共同估算是共同跟进的基础，在**每日立会**中人们之所以关注别人的进展，就是因为人们关心自己曾经参与的估算结果是否正确，方案是否可行。

☺共同估算和每日立会产生了**同行压力**，即每个人都希望以好于或至少持平于团队估算的结果完成任务，从而产生了**受激励的个体**。

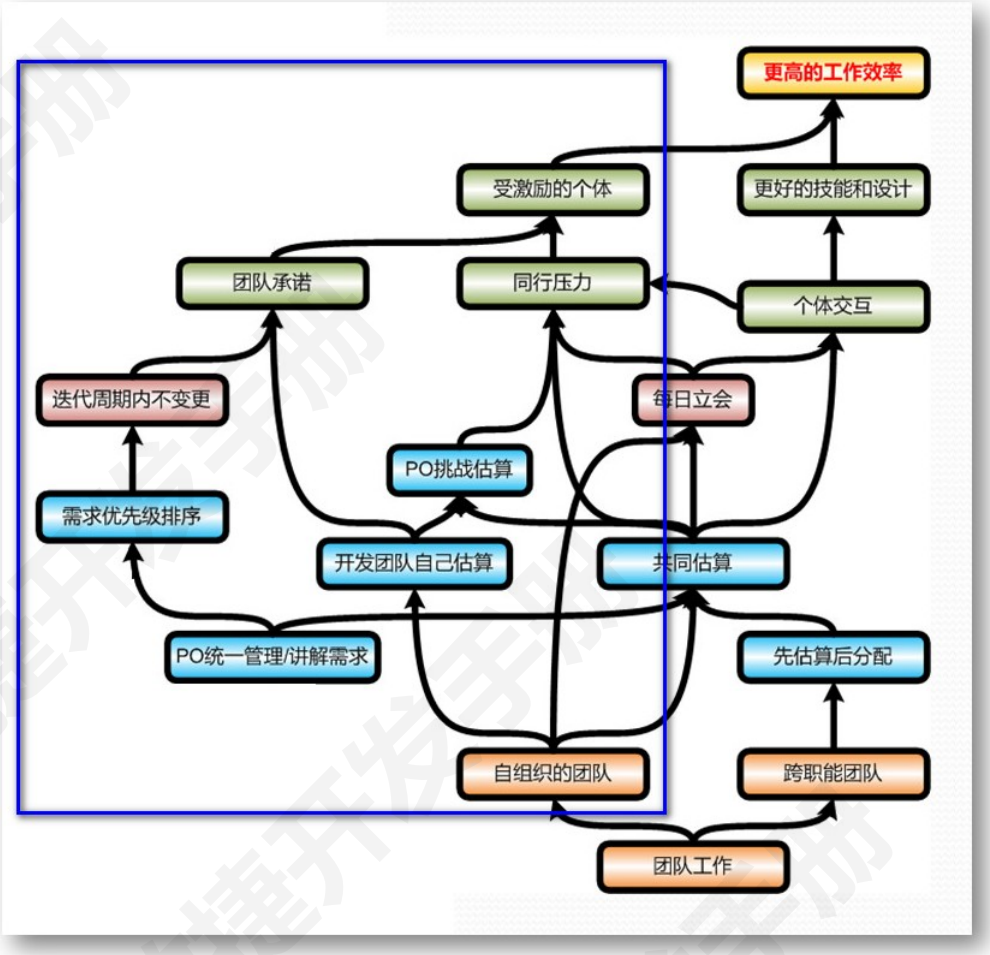
☺共同估算和每日立会还防止个体失误，前者通过统一了大家对同一个需求的理解和其实现方法的理解，后者则防止了工作中出现需求镀金、蛮干等问题，从而产生**更好的技能和设计**。

☺作为**自组织团队**，敏捷团队并非简单地因为“领导让我们自我管理”而受到激励，而是在跨职能团队、共同估算、个体交互、同行压力这些实践的配合下才产生了受激励的个体和更好的技能和设计，从而产生**更好的工作效率**。

更多分析，请见：敏捷开发生态系统系列之二：敏捷生态系统-计划跟踪 I（跨职能团队-共同估算-每日立会-同行压力）

回目录

# 敏捷生态系统-计划跟踪 III




## 需求优先级排序-迭代期内无变更-团队承诺

☺产品负责人（PO）与团队的正确互动是**自组织团队**正常运转的核心机制之一。

☺产品负责人的权利是**统一管理和讲解需求**以及**需求优先级排序**，而义务则是接受开发团队的估算，并承诺**迭代期内不变更**。

☺团队的权利在于**开发人员自己估算**，而义务则是接受产品负责人所设定的需求内容、实现标准、优先级排序，并对自己的估算做出**团队承诺**，这种承诺会造成整个团队受到激励。

## 自组织团队-开发团队自己估算-PO挑战估算-同行压力

☺产品负责人不能干预团队的估算，却可以**挑战估算**。挑战估算可以通过对比两个团队处理同类任务、对比以往同类任务与本次任务等方式进行。团队的荣誉感会令团队产生团队间的**同行压力**（与其他团队及与自己的过去相比），并进而产生**受激励的个体**。

☺作为**自组织团队**，产品负责人与团队互相没有领导关系，却通过分权与承诺管理，使得两者互相管理，从而产生**更高的工作效率**。

更多分析，请见：敏捷开发生态系统系列之三：计划跟踪II（需求优先级排序-迭代期内无变更-团队承诺）

以及：敏捷开发生态系统系列之四：计划跟踪II（自组织团队-开发团队自己估算-PO挑战估算-同行压力）

回目录

# 敏捷绩效管理-考核对象的变化

## 传统绩效考核面向个人

利用个人绩效的压力，来产生主动工作的动力。

- ☻ 队员会倾向于独自工作，忙的忙死闲的闲死，工作成果受到个人瓶颈制约
- ☻ 在需要同时多人合作或前后衔接的场景中，容易引发部门矛盾
- ☻ 个体能力有限时，解决问题的能力亦受限
- ☻ 团队关系冷漠，新人得不到帮助，无法成长
- ☻ .....

## 敏捷绩效考核面向团队

为整个团队设定目标并整体考核，利用同行压力来产生工作动力。优点：

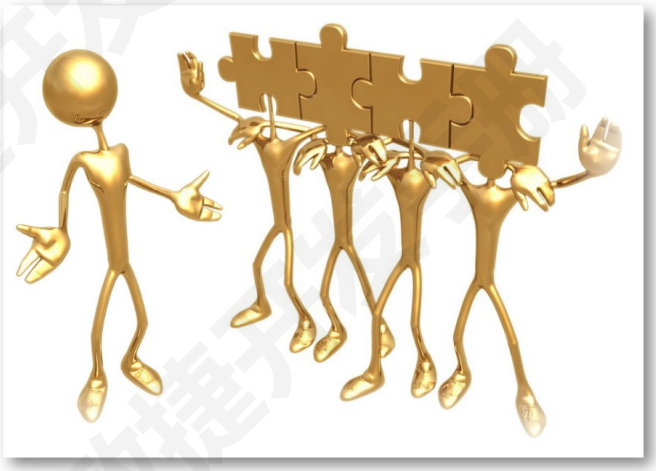
☺团队作为整体工作，消除了能力和资源利用的瓶颈

☺团队会依据自身情况，自行选择以何种方式达到最大生产力

☺各自为战？高手工作大家帮忙打杂？大家工作高手巡回帮助？.....

☺团队关系融洽，新人不断成长，形成学习型团队

☺.....



更多分析，请见：敏捷开发绩效管理之一：序言及“敏捷开发是否考核个人”（绩效考核）

以及：敏捷开发绩效管理之三：个体动力之源——同行压力（松结对编程，师徒制度，跨职能团队，绩效考核）

回目录

# 敏捷绩效管理-为团队设定目标，让团队把控细节

## 传统管理方式：领导指派任务

领导事无巨细地分配任务，追踪任务

- ☻ 团队缺少自我思考能力，丧失主观能动性
- ☻ 在估算问题上博弈，团队加一倍，领导砍一半
- ☻ 推一步走一步，领导累，经理累，队员也累
- ☻ .....

## 敏捷管理方式：自组织团队

为团队指明整体目标（如Sprint Backlog），具体工作细节（如所用技术、人员分配）由团队自己决定。

☺团队在计划会上会群策群力寻找最佳方案

☺团队会努力将自己估算变为现实，达成承诺

☺团队依据日常工作中的实际情况调整工作，无需依赖领导指令

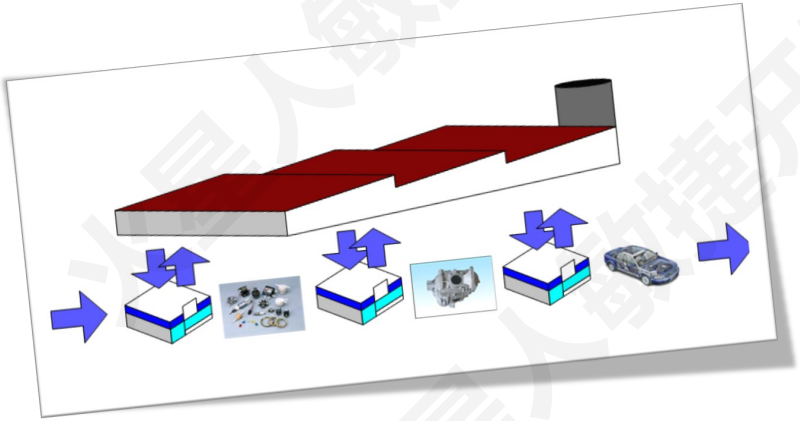
☺.....

更多分析，请见：敏捷开发绩效管理之四：为团队设立外部绩效目标（目标管理，外向型绩效）

以及：敏捷开发绩效管理之二：用中医理论管理团队及其绩效（绩效考核，团队管理，自组织团队）

回目录

# 智慧敏捷-精益生产的启示



**传统工厂的管理方式：中间产品众多**

工厂：

- ☻ 大量资金积压在不可销售的中间产品上
- ☻ 在仓库内存储及来回搬运的人工费用很高

软件：

- ☻ 大量人力花费在不可销售的中间文档上
- ☻ 编写文档的速度很慢，评审的效率很低，而读取过程中很容易产生歧义

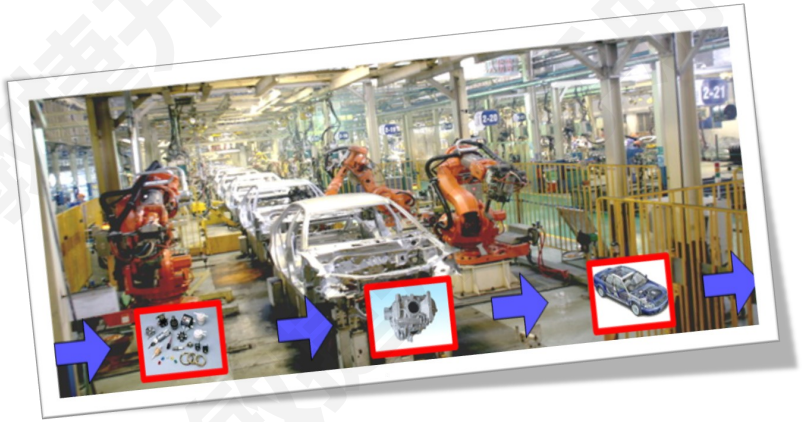
**精益生产认为：若能减少浪费，则等同于提高提高生产率**

工厂：

- ☺ 若不再经过零部件环节，则可以节省大量资金
- ☺ 若不需要仓库，则可以节省大量来回搬运的人工费用

软件：

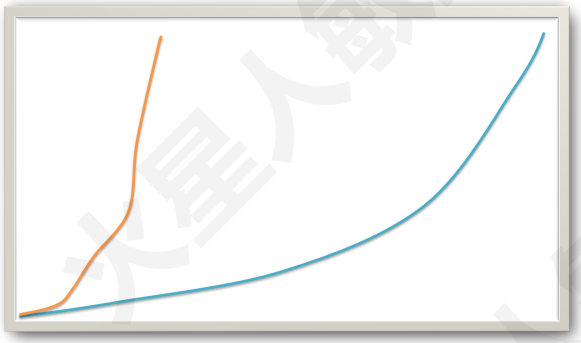
- ☺ 若尽量减少不可交付的文档，则可以节省大量人力
- ☺ 若以跨职能团队尽量减少中间交接环节，则可以有效减少文档的数量，进而提升生产效率



回目录

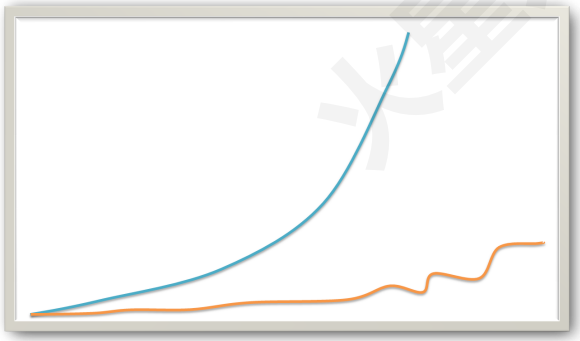
# 智慧敏捷-写不写文档？

## 过度设计和返工是造成时间浪费的主因，不同类型项目中两者比重不同



大型项目尤其是系统工程级别的项目，比如军工、航空类项目，**设计**的工作量很大，原因是这种投入毕竟是可控的；而一旦由于**设计工作不充分**，导致严重的**返工**，则往往不是简单的费用问题，还往往造成项目被终止。

因而在这类项目中推广敏捷，应该适当增加文档的数量，以便长期项目能够按计划完成。



在互联网、消费电子行业则正好相反，**返工**主要是由于**业务变化**而不是错误或不足的设计引起的；相反**过度设计**往往在未被付诸实现之前就已经过时，反而形成浪费。

因此在这类项目中推广敏捷，应当适当降低文档的数量，以便在业务变化时轻装上阵，而不需要同步修改大量设计文档。

应当理解敏捷开发的出发点不是不写文档之写代码，而是**减少浪费**，以便能按照自己项目的特点，灵活选择文档的数量及其形式，在过度设计和返工之间找到平衡。

更多分析，请见：智慧敏捷系列之一：序言，之二：写不写文档，之三：做不做架构设计

回目录

# 智慧敏捷-敏捷实践的表象与内涵

## 每日立会开多久？

甲：“我们每日立会开不起来。”

乙：“嘿，我们每日立会开起来了，而且越开越长了，一开就是1个小时，净是些技术细节。”

甲：“别人等着他们讨论，那多耽误时间啊……”

乙：“我也觉得是，但是敏捷开发鼓励面对面沟通，到底应该打断还是不打断呢……”

看似沟通顺畅，但是如果队员们把每日立会当作唯一的沟通机会，甚至用来解决技术问题，可能表明他们在会下极少沟通。

## 定不定规范和模板？

甲：“你们的设计文档打算怎么写？”

乙：“到时候再说。”

甲：“应该有规范的开发流程和模板，才能写好设计文档。”

乙：“预先定义的流程和模板未必适用，敏捷开发崇尚推迟决策，只有在具体工作之前才能决定是否写，怎么写最好（maximizing the amount of work **not done**）。”

甲：“你们组才3个人，能决策出比组织级定义的流程和模板还好的吗？”

在过多的流程制度制约了开发灵活性的同时，完全没有规范和模板也可能造成团队无所适从，反而浪费时间。因此敏捷开发需要在“个体与交互 和 流程与工具”之间，找到平衡点，而并非完全放弃流程与工具。

**智慧敏捷：敏捷开发应该注重各种实践的内涵，而不要被表象欺骗。**

更多分析，请见：智慧敏捷系列之三：每日立会开多久，之四：定不定流程和模板。

回目录

# 中英文对照词汇表


回目录

# 版权及使用授权说明



# 使用方法

- ☺ 作为培训前的预习阅读。
- ☺ 打印并张贴在公司走廊上。
- ☺ 作为企业内部小组培训教材使用。

如有意见和建议，请访问作者博客并在讨论页面中参与讨论：www.cheny.com。