公司介绍

北京迅思威尔科技有限公司 （AgileDo），国内专业敏捷项目管理和敏捷开发过程的培训、实践基地。

迅思威尔是由一批来自国际著名 IT 公司的软件精英、海外留学人员组成。公司致力于软件敏捷开发领域，特别在 Scrum 领域有多年的实践经验。

公司拥有敏捷开发专业顾问团队。所有咨询顾问均是有 10 年以上 IT 企业的从业经验者，对软件开发、项目管理等相关领域有精辟的理解。他们来自于国内外顶级软件公司敏捷开发的实践前沿，带来了国外同行成功的经验和国内团队成员自身敏捷实践的成功经验。一直以来，与各国资深敏捷专家密切的联系，定期的面对面沟通，使迅思威尔敏捷理论和实践保持业内的前沿。

迅思威尔专注于敏捷培训、敏捷咨询（特别在 Scrum 培训、Scrum 咨询）。

作为中国软件行业协会系统与软件过程改进分会的重要一员，迅思威尔愿与企业分享国内、国外敏捷相关的经验和教训。

公司提供的服务

通过公开培训课程，您可以：

了解企业级实施敏捷模式、Scrum 项目周期管理、外包项目实施敏捷实践、小企业实施 Scrum 实践、以及 IT 企业研发管理解决方案。

我们的特点是：以案例为主线，70%具体实施实践，30%理论

通过企业内训课程，您可以：

根据企业的现状定制感兴趣的敏捷开发主题，培训课程对定制主题做更深入的分析和更细化的实施措施

通过咨询服务，您可以：

咨询顾问和您一起工作，发现企业问题的根源，提出具体的解决方案，并和您共同实施。

如何联系迅思威尔（Agiledo）

邮件：andy@agiledo.cn MSN：agiledo@hotmail.com 电话：(010) 59454508 网址：http://www.agiledo.cn/

作者简介

Andy Yuan （袁斌），迅思威尔咨询顾问，10 余年中就职于全球性公司从事软件和产品的开发。曾任 Anoto 产品中国区开发总监和 Mino 亚洲区软件开发总监，利用 Scrum 成功实现产品快速交付，在 onshore/offshore 的 Scrum 开发，Scrum 团队建设，Scrum 在小型、大型团队的应用等方面积累了丰富的经验。文章中的实践内容来源于作者以往项目的 Scrum 实际案例。

文档的内容：（**所有内容的文档为 30M，81 页，蓝色内容以及所有的图例，不在此精简版的文档范围内，可以通过公开课、培训、俱乐部、沙龙等面对面方式交流**）

图例：

：初稿完成； ：终稿完成； ：未开始； ：进行中

Scrum 实践的内容包括：我们团队如何实践 Scrum 过程、经验教训（重点）；Scrum 实践的典型问题（重点，包括我们实践中遇到的以及网络中被大家重视的）；一些 Scrum 推荐的理论（非重点，我只是摘录了一些书籍中的文字，例如《Scrum - Checklists》，不系统也较肤浅，最初目的只是为了让自己有一个比较而已。关于 Scrum 的理论，我自己做了两个 PPT：Scrum、Beyond Scrum，在团队内部沟通是这两个 PPT 来进行的）

Scrum 实践过程中参考了其它敏捷的开发方法：Agile Modeling & XP

Scrum 实践过程中参考了很多同行的实践做法以及理论文章，在此深表感谢。如果由于我的疏忽在文中没有标注所有参考文档的原作者出处，请见谅！
特别感谢：Mike Cohn：《agile estimating and planning》；Boris Gloger：《Scrum - Checklists》；Esther、Diana《agile retrospectives》；Robert C. Martin《敏捷软件开发：原则、模式与实践》、InfoQ 网站

Scrum 实践还在不断总结中。已经了解，正在融入实践的部分：KanBan、Lean。在后续的版本中会共享我们这方面的实践。

实践 Scrum 的思路：参考了 Henrik Kniberg 推荐的 Scrum 开发模式
（具体开发模式图见“我们这样实践 Scrum－201003-完整版”）

目　录

图 例

团队建设

推荐理论

1.Cross-functional 团队

2.Self-Organization

3. 团队尽一切可能去完成任务——发布产品。团队需要全面的能力，这意味小组内拥有实现产品的全部技术和技能。团队还需要充分的理解产品负责人所描述的产品愿景以及 Sprint 目标，以更好地支持可能需要进一步开发的产品的发布

4. 敏捷团队，QA 的特点：

作为开发团队的一员，和整个团队一起对质量负责：

- 关注编码，例如评审设计，（对技能要求更高）
- 自动化测试（对技能要求更高）
- 在客户很难和 Team 工作在一起时，QA 更多承担 customer proxy
- 程序员同样关注质量，特别在代码的质量，包括是否符合需求，是否可以自动测试...

图（典型的敏捷团队的 QA 职责需求）展示了一个典型的敏捷团队的 QA 职责需求。

5. 充分的授权是指事先已经给团队提供了完善的支持－训练、信息、资源，然后让团队能够自主地做出正确的决定。否则的话，让团队做决定等于就是弃他于不顾

6. 作为PM/SM，你如果完全放任，可能也就不需要你了。授权的关键在于使人们有权做他们自己的工作，并持续考量他们的工作结果，而不是置身局外

7. 自管理团队演化的过程：

Zawacki and Norman 认为成功的自我管理团队由以下 5 个阶段不断演化而成：

- 阶段 1：传统的等级体系的组织结构，领导对下属进行一对一的管理和指导
- 阶段 2：领导转变为管理者（group manager），角色逐步向团队的协调者或者教练转变
- 阶段 3：管理者变成了团队的协调人，并向团队成员提供必要的培训使其承担更多的领导任务
- 阶段 4：团队承担有管理者承担的责任，管理者变成了团队外部接口
- 阶段 5：团队的管理者是团队的其中一个资源

8. 自我管理的团队并不意味着由团队来决定实现的目标，这是由 PO、客户决定的。自我管理的团队决定如何适应环境，而管理者会影响环境.在实践中往往会听到团队说“这件事我们做不了，所以我们决定不做”…

9. 高效的自我管理团队应有以下特征：

- 清晰的目标。自我管理团队对所要达到的目标有清醒的了解，激励着团队成员紧密地凝聚在一起，形成一个真实的“命运共同体”。

- 相关的技能组合。这些团队成员拥有很多互补的技术和技能，并通过不断学习新的技术和技能来增加他们技术和技能的多样性、灵活性以及对于团队的价值。
- 共同的承诺。
- 良好的沟通。自我管理团队具有一种氛围可以让成员之间相互倾听并畅所欲言，有利于培养创造性与风险承担意识。团队成员直面和公开处理不同建议，并做出真诚的反馈。他们了解彼此之间工作的进展程度，形成合力。
- 恰当的领导。自我管理团队的领导者往往担任的是教练和后盾的角色，对团队提供指导和支持，但并不试图去控制它。
- 环境支持。从内部环境来看，团队应拥有一个合理的支持体系，包括适当的培训，一套易于理解的用以评估员工总体绩效的测评系统，以及一个配套的人力资源系统。这种支持体系能保证并强化成员行为以取得高绩效水平。从外部环境看，团队可以从多种渠道获得完成工作所必需的各种资源。

我们的实践

1. 角色：高级软件工程师、架构师、美工、QA、第三方技术支持

2. 最关键的特性：必要的技能、沟通意愿、对结果负责

3. 我们尝试了团队完全由高级软件工程师组成，质量由工程师负责，结果非常不好，一个是质量差，另一个是没有很好的技术高手，Coding Review、设计必要的前瞻性和技术难点经常困扰。后来增加了架构师，增加了编码流程，部分解决了第二个问题和第一个问题中的代码质量，但功能质量还是不能保证，例如边界、各种异常、非关键的正常流程往往被忽略。于是增加了 QA。此时的流程是：Daily Build，开发人员根据功能最主要的验收测试后发布在测试平台上，由 QA 进行测试。这样 QA 做 Story 增量测试，同时还可以兼任部署的工作。但我们还是有问题无法解决，例如 CMS、Ajax 技术难点、整体网站优化，这样需要更加专业的技术服务，我们和一些专业公司签署技术服务协议，包括外包、现场支持等。美工是我们最后加的，是部门的资源，在需要的时候申请**天全职在团队中。需要说明的是，这里的“架构师”其实是某一个专职的架构师和团队内资深的程序员组成的一个小组。

4. 最核心的是人，所以选人是第一步的，如果已经入职的人员不合适，磨合仍然不能符合团队要求，尽早让其离开团队，调整到公司的其它岗位。

5. 如果员工的新点子能够被团队接受，那就实施，这是非常有效的非物质激励的方法

6. 在团队刚开始建立的时候建立“如何一起工作”：有明确的完成标准、互相尊重、鼓励沟通、如何开 Daily Meeting、如果做 Sprint Planning 等等

7. 架构师的职责

在敏捷团队中，架构师的角色和传统开发团队中架构师的角色有很多重复的地方，也有一些不同的地方，我们团队的架构师，工作如下：

- 在 Sprint 之前，按照 Agile Modeling 的思路建立系统的架构（不是重量级的），这个架构是和团队（主要指 Senior developers）一起完成的，并欢迎团队中每一个人为解决方案做出贡献。
- 在 Sprint 中，负责具体的重构（架构和关键业务逻辑）
- 在 Sprint 中，关注代码的质量和性能，具体是在 Coding Review 时及时提出问题并提出解决方案，同时他的编码会作为小组代码规范而统一；同时通过 Jmeter 进行压力测试，以判断性能情况。
- 在 Sprint 中，是一个指导者和协调者，可以确保团队不会因为缺少某些经验或是对某些技术感到不适而简单地拒绝一种设计方案。也可以在团队中经常出现不同意见。还会带来激烈的争吵时来帮助团队打破僵局，重新和谐。
- 在 Sprint 中，和 PM 一起从业务价值的角度解释 Team 用到的技术方案的选择，所做的技术方案绝不是为了体现技术领先，而是在成熟技术和业务价值之间做平衡。
- 在 Sprint 中，Story 的实现不是工作重点，但会承担部分 Story 的实现

8. 如何让团队保持热情：

除了绩效考核外，还有一些事情可以使团队保持热情，我们是这样做的：

- 确定迭代周期，不至于太短使团队压力过大，也不至于太长使团队松懈。工作首先是热情的一个开始，我们的迭代周期是 3 周
- 让大家每个人的工作放在整个团队的面前，例如需求细化、设计、代码、个人进度，这些都放在 Stand-up Meeting 中。有些功能工作量大，可以 3 天有一次 meeting，遇到紧急或者需要大量讨论的功能，可能半天或者随时就有一次 meeting。
- 说真话必须受到保护。团队成员提出的建议，只要是对团队有用的，尽可能去实施，这个比任何口头宣传的“尊重、信任”都更有效
- 让市场的反馈经常传达到团队，无论是好的评论还是坏的评论，都会对团队的工作进行触动。成就感和危机感都会使团队在一种工作状态
- 给团队成员以技术方面的培训。这里的技术培训，我们一方面是让团队自己去摸索，另一方面将很难的技术问题外包，得到经过论证的可行的技术方案，再应用到团队中。这样团队学的更快，对产品的风险也更小
- 要保持 SM/PM 的热情，如果他懈怠了，在团队还没有成为真正自我管理团队之前，一段时期内整个团队的节奏都会慢下来，SM/PM 在使以上措施得以实施的过程中起到催化剂的作用
- 让及时推动也无法成为“Pig”的成员早一些离开团队

9. 团队要对承诺负责

SM 的角色

推荐理论

Scrum Master 是整个团队的导师和组织者，他负责提高团队的开发效率。他常提出培训团队的计划，列出障碍 Backlog。Scrum Master 控制着检查和改进 Scrum 的周期，他维护这一团队的正常运行，并与产品负责人一起让利益相关方获得最大化投资回报。他关心的是这些敏捷开发思想是否能得到利益相关方的理解和支持。
例如：
1. ScrumMaster 需要知道什么任务已经完成，哪些任务已经开始，哪些新的任务已发现，和哪些估计可能已经发生变化。ScrumMaster 需要根据以上的情况更新反映每天完成的工作量以及还有多少没有完成的 Burndown Chart。 ScrumMaster 还必须仔细考虑进展中的开放任务数，进展中的工作需要得到最小化，以实现精益生产率的收益。
2. ScrumMaster 需要找出阻碍 Scrum 的障碍和依赖。他们需要优先次序和跟踪。根据优先级指定计划解决这些障碍。其中有些问题可以在团队内部解决，有些则要团队之间的协调，还有的要管理层的介入来解决，甚至有些是公司的问题阻碍了团队达到他们的生产力。

我们的实践

1. 和 PO 讨论 PB 中的优先级。SM 讨论时要首先了解 Product Vision、Product Roadmap 和 Business Value，以及更主动的和客户沟通，以了解客户的真实意图以及客户环境的制约和限制。有时 PO 只是为了增加市场机会，有时 PO 自己也没有想清楚。需要 SM 和 Team 去提出更多的问题去澄清。有一个例子：产品功能中要求不能在凌晨发送短信，但只有电力行业例外。PO 提出了一个 User Story：系统管理员可以定制每个用户何时可以发短信，认为是高优先级。我们（Scrum Master 和 Team）在评估 business Value 时提出问题：电力行业客户是否是我们的目标市场？公司是否有资源做电力行业的销售？经过讨论后，PO 的用意是来自 marketing 部门，他们不愿意放过任何一个可能的市场机会，但 marketing 也承认确实机会比较小。这样，这个 User Story 的优先级非常低。

2. 在团队还不能做到自我管理(committed, proactive, collaborative)时，SM 要权衡领导和管理的比重。我们最开始实施 Scrum 时，两者的比重是 4:6,然后是逐渐倾向于领导的比重越来越多。

3. 如何使敏捷开发思想是否能得到利益相关方的理解和支持，我们这样和大家宣扬 Scrum 的好处：
a、对公司的影响：公司的各级可能提出需求的部门（老板、市场部、销售、产品部、客户部等）不再是想到一个新的需求，得到客户的一个反馈，就给研发部发布需求，而是公司有了一个机会，总结这些需求，分析业务的优先级，从 business value 的角度安排公司的资源，提高 ROI。

b、对公司的影响：对市场更快的反应速度，少的，核心的功能，保证了有市场功效的快速发布，增强公司的市场竞争力。特别是对于新成立的公司或者是新产品的研发。
c、对团队的影响：持续的，可见的成就感。同时，可以很好的和公司的其他部门以及客户建立信任关系。定期的 working system 发布，每日的 stand meeting，随时的 face to face 沟通，任何项目的 stakeholders 都可以随时了解项目的进展，信任也会很快建立。当然，前提是需要高质量的发布。
d、对个人的影响：快速的学习机会和环境。cross-functional 的团队环境，对于每个人都有更多的学习机会。同时，每个人都有发言权，会更长时间的延续工作的热情。

4. 如何使敏捷开发思想是否能得到利益相关方的理解和支持，我们这样沟通：
a. 首先要站在对方的角度说话。例如和管理层沟通，就要站在公司的角度讨论问题。和 PO 讨论问题，就不要直接说某某功能不能做，因为最终的优先级的决定权在 PO，可以先了解需求，讨论一下可能的工作量，介绍团队的资源，以谈判的方式沟通，而不是先造成潜在的对立环境。
b. 维护自己的利益。软件的质量是由 Team 负责的，如果无限制修改需求，则会导致质量的严重下降。曾经遇到这样的 PO：我不管开发团队是否可以完成，反正我已经提了需求，不能完成是开发团队的事情。这个时候就需要和管理层沟通
c. 用数据说话，数据可以有一些 Buffer，但要基本靠谱。“这个功能没有时间做”，这样的沟通只会造成对立情绪。如果换成“这个功能需要 20MD，整个 Team 可以使用的资源只有 30MD，而这个功能的优先级也很低，增加了这个功能就不能完成更高优先级的**功能，PO 你可以决定是否增加这个功能？”

5. SM 如何做整个团队的导师和组织者
a、和团队一起工作
和团队工作在一起。每一个 ScrumMaster 的背景不一样，有偏技术的，有偏管理的，网络上有各种各样的讨论，有人认为 SM 不应该涉足 Sprint 内的任务（Developer），而要把重心放在指导、沟通、保护团队上；有人（例如 Mike Cohn）认为 SM 可以一定程度上做一些 Developer，但实现的任务一定不能成为关键路径；还有人认为 SM 需要成为 Core Developer，以保证任务的实现，特别是技术难点或者成为救火队员。我自己很有意思，正好是头 6 年偏重实际的编码、设计，后 6 年偏重管理（project management、people management）。我是这样做的：指导、沟通、保护团队会占据 70%的时间，技术讨论会占据 30%的时间。我不会参与编码，也不会做救火队员，不是我不想编码，而是我认为所有的对工作量的评估，应该由团队通过自己的实现来形成自己的感觉。我做的具体工作有一些例子：所有的需求我会跟踪到细节；技术设计讨论我会参加，会发言，但不会主导，由 Team 主导并形成最后决定；代码质量交由团队 coding review 以及静态代码检测工具（findBug）等；测试我会自己将功能完整过一遍；做燃尽图；…
b、以 Review 为主。由团队提出方案，SM 提出问题，而不是直接提出方案。
c、培训，更多的是以非正式的方式和言传身教的方式
d、对团队已经认可的工作方式，要在团队中固化，并不断改进。这一点非常重要，但很容易被忽视

PO 的角色

推荐理论

产品负责人是利益相关方的代表，他的工作重点是产品的业务方面。他负责向团队介绍产品远景。他负责给出一份明确的，可度量的，合理的产品 Backlog，并从业务角度出发对 Backlog 中各项问题按优先级排序

我们的实践

1. 我们对 PO 的要求：全职、有深厚业务背景、有很好的沟通技巧可以代表利益相关方排列优先级。我们目前一个人不够，有两个 PO 为这个项目服务，但对于团队而言，听到的是统一的意见
2. 最初是 SM 帮助 PO 做出最原始的 Product Backlog，以供前几个 Sprint 使用。PO 也是逐渐适应 Scrum

Product Backlog

推荐理论

Mike Cohn 推荐了如下的因素来评估优先级：

1. 经济价值、客户的意愿
2. 开发成本
3. 给公司和团队带来的经验和知识
4. 消除的风险

经济价值主要来源于增加收入以及减少成本。如果经济价值不容易估算，那么用客户的意愿来判断吧：）

增加的收入包含：

- 新领域（客户）带来的收入
- 原有客户新增购买量带来的收入
- 可以独立销售的模块、配件带来的收入
- 被市场接受更高价格的功能带来的收入
- 衍生服务带来的收入

减少的成本包含：

- 保留客户的成本（如果功能不开发，将失去老客户）
- 可能的效率低下造成的成本，例如培训新员工，部门间的沟通、过长的开发周期

客户的意愿

关于“客户的意愿/满意度”，推荐 Kano 模型：

- Threshold：必须包含的功能，是一个客户接受的门槛
- Linear：功能的质量、性能、易用性等指标越高，客户越满意。
- Excited：可以给客户带来惊喜，但没有这些功能，也不会对满意度有负面的影响

给公司和团队带来的经验和知识，包含产品级的和项目级的。产品级的知识指通过某个新功能的开发来了解产品应该不含哪些功能以及不应该包含哪些功能，例如某个功能在市场上的“投石问路”；项目级的知识指产品如何被开发出来，例如使用的技术、人员的技能、团队如何工作等等。

风险主要包含“计划风险”、“成本风险”、“资源风险”、“技术风险”、“商业风险”等等

我们的观点是：

1. 以价值（#1 和#3）和风险（#4）作为第一层面的考量，是“应该做”的层面。优先级的顺序如图（价值－风险：优先级的判断依据）所示
2. 以开发成本（#2）作为第二层面的考量，是“能否做”的层面。以这个为参考，适当调整“应该做”。

部分内容摘自 Mike Cohn：《agile estimating and planning》

我们的实践

1. 组成：功能性需求、非功能性需求、Bug
2. 形式：Story（Who、What、Why）
3. 如何考虑优先级：

3.1 PB 分为三类：最重要的，重要的，一般的。

最重要的指必须要在最近的 Sprint 内完成的，例如客户明确提出的、公司管理层提出的，系统有明显漏洞的，就算团队资源不能满足，那就必须拆分功能；

重要的是指尽可能在最近的 Sprint 内完成，例如客户不确定的、管理层不确定的、上个版本的 Known Issues，如果团队资源可以满足最重要的功能之前还有余力，优先考虑此部分需求。一般的是指在 2～3 个以后的 Sprint 考虑的需求

最重要的（> 200），重要的（100 ～ 200），一般的（1 ～ 99）。数字越大，优先级越高。

4. 开发团队如何评判 Product Backlog 中的 Item?

我们曾经用到的一些方法/原则：

- 产品的设计是否对用户的使用增加了规则和约束。如果是，那么就具有很高的风险，因为用户是不会按照我们的思路出牌的，而且我们也不知道用户实际的使用环境。
- 产品的功能是 PO 提出的，还是最终用户提出的。如果只是 PO 提出的，是需要和最终用户沟通后再加入到 Backlog。Team 一般会做出原型，以得到客户的确认。
- 特别关注正常流程和异常流程，以及非功能需求。细节是最好的讨论线索。
- 要求给出用户使用场景（User Scenario）
- 涉及用户的体验，原则就是“最小的付出，得到最大的收获”

5. 我们如何管理需求变更

- 我们用 Product Backlog（PBL）记录产品需要完成的功能，此功能包含功能需求、非功能需求（性能、高可用性、可扩展性等）、上一个 Sprint 发现的 Bug
- 每一个需求会根据业务优先级加入到 PBL 中
- 优先级越高的，需求讨论的越细；反之，需求讨论的越粗
- PBL 中的需求会经常被重新根据优先级排序，甚至被删除例如有新的需求加入。
- 每一个 Sprint，只做优先级最高的，Team Velocity 可以接受的需求

具体可以参考图（我们如何管理需求的变化）

说明：

由于我们公司是做电信增值业务，产品有设计、上线、运营等阶段，根据我们的实践，优先级是这样的：

- 在产品上线前，来自移动的要求、产品的关键卖点、系统的可扩展性的优先级最高
- 在产品的试运行期间，客户的反馈、性能、高可用性的优先级最高
- 在产品运营阶段，系统的监控、报警的优先级最高

我们的 PO 会接到各个部门的不断增加或者变化的需求要求，但以上优先级的方式是管理需求变更的原则

6. 我们如何写用户故事

- 选择一个好的工具，这样和用户沟通时可以很容易做头脑风暴以及方便记录，我们选择的是免费工具：FreeMind
- 每一个故事有三个要素：角色、事件、原因，我们把角色作为二级目录，事件及原因作为三级目录，分解的子事件作为四、五级目录，最后目录中的最后一级叶子和二级目录中的角色构成了“故事”。 “故事”不但包含“用户故事”，还包含“开发故事”
- 为每一个“故事”写验收测试，具体就是分为不同的用户场景，每一个场景的描述为“当…的时候，做了…，会得到…”

好的故事应该是：

独立的，有价值的，可评估完成时间的，清晰并可以很快完成的，可以被测试的

我们有一个实际例子，参考图（我们这样写做用户故事）。其中，蓝色的文字部分是 Sprint Backlog，每一个分支的叶子部分是最后分解的“故事”，或者说相对独立的小功能。红色的文字是其中一个功能分解为实现的任务。

评估会议

推荐理论

1. 参加者：PO、SM、Team
2. 前提：PB（Story）已经排列好优先级，故事点评估卡片（0，1，2，3，5，8，13，21，34，89）为每一个团队成员准备好。当然，Ideal day 也是另外一种评估方法
3. 过程：PO 介绍需要评估的 Story 背后详细的用例，各成员首先选择最小用例的 Story X，指派其工作量为 1，然后针对每一个 Story 和 Story X 的工作量比较，同时亮出评分卡片。如果有分歧，分歧最大的成员进行讨论，然后再次投票，知道所有人意见一致为止。将评估结果添加到 Backlog 中。
4. 交付物：更新后的 PB 通知所有参加会议的人员
5. 持续时间：无推荐

我们的实践

1. 参加者：PO、SM/PM、Team
2. 前提：PB 已经准备好，可以是故事，也可以是 Feature List，但往往 PO 不会提供 Why，SM、Team 也不追究，可以要求 PO 提供需求来自哪里（直接客户、老板、市场/销售部门、产品部、技术部门），这个在现实中成为优先级的排列依据。PB 的 Item 没有任何书面的细节。PB 包含了业务需求、上一个 Sprint 遗留的 Known Issues、技术部门提出的技术需求（开发环境的搭建、基础架构的实现等）
3. 过程：PO 介绍需要评估的所有 Story 背后的用例，仅是 High-Level，主要的正常流程即可。各成员首先选择最小用例的 Story X，指派其工作量为 1，然后针对每一个 Story 和 Story X 的工作量比较，我们是用 IMD 为评估方法，直接用任何近似的比例，例如可以认为是 4.3,也可以认为是 2.7（我们也在同时尝试用评估卡片的数值，正在看不同的效果）。大家同时亮出估算比例，分歧最大的成员进行讨论。对于任何和现有功能依赖关系很弱的功能或者团队成员对整个系统都很熟悉，最多讨论两轮，根据最后所有人的平均值作为估算比例，这些功能可以由团队中的任何人实施；对于团队中少数人了解的功能或者技能（双机热备等），主要由这些人进行评估，并给出理由，其他人给出一些辅助意见，同时这些功能也主要由这些人完成。大家评估最小用例 Story 的 IMD，然后按照估算比例计算其他故事的 IMD。最后将估算结果添加到 Backlog 中。如果 PO 先介绍完 Story，成员已经比较清楚，工作量的评估 PO 可以不参加。参加的好处是透明的过程增加双方的信任感。用 IMD 的最大原因是 PO、管理层需要明确的估算时间
4. PO 形成期望的 Sprint 目标，这个目标是根据业务优先级以及在评估会议上得到的 Backlog 的评估值由 PO 综合决定。此时 PO 提供期待 Sprint 目标中 Backlog 足够的信息，并以文档的形式发给团队。例如报表需求，足够的信息包括，通过哪些查询条件，得到什么查询结果，查询条件和查询结果中的字段定义

5. 交付物：更新后的 PB 通知所有参加会议的人员
6. 持续时间：4~8 小时

Sprint 计划会议 1

推荐理论

1. 明确 Sprint 的时间表

选择迭代周期，一般有以下一些方法：

- 根据整个 Release 的周期，一般至少需要 4～5 个迭代以获得客户的反馈
- 根据不确定性的数量，一般不确定性的数量越多，迭代周期越短。最需要关注的不确定性包括：客户到底需要什么？每一次迭代 Team 到底可以完成多少？项目的技术风险？举例：如果我们想增加一个高优先级的新增需求，那么这个需求一般要在 1.5 个迭代之后才可以提交。见图（增加一个新的需求何时可以发布）
- 根据每一次迭代的成本。成本越高，迭代的周期越长。全用例测试的时间和资源是最典型的迭代的成本。
- 根据团队对压力的感受。好的开发节奏，应该是团队感到压力比较大，但努力一下可以达到。团队没有压力或者压力非常大都不是好的开发节奏，时间长了会严重影响士气或者产品质量。假如一个团队在 4 周的迭代周期中，第一周开始仍然比较松散，那就要考量一下是计划不合理，还是适当缩短迭代周期。

2. PO 向团队阐述产品的远景
3. 如果 Backlog 中有问题遗漏，PO 可以向 Backlog 中增加问题，并重新排列优先级
4. PO 和团队一起确定 Sprint 目标
5. 持续时间：4~6 小时

我们的实践

1. PO 明确 Sprint 的时间表，这个一般确定在 4 周，但是也会因为客户的要求或者市场的要求而改变。例如我们的客户一般要求每月的第一天开始部署新的程序，此时我们的 Sprint 周期为 1 个自然月；如果我们的市场宣传需要一个配合（20 天后），那么 sprint 周期会定在 20 天。目标只有一个：提供价值
2. PO 会向团队介绍半年之内的 Roadmap，这个团队非常关心，因为可能涉及到设计等；对于产品的远景，团队会听一听，但不会特别关心，有时候 PO 就不会涉及产品的远景。
3. 如果有的话，PO 或者开发团队增加认为遗漏的问题，并调整优先级。如果这些问题有在 PO 期望的 Sprint 目标内的，团队需要评估工作量，否则可以留在下一次的评估会议再评估。

以上持续时间：1 小时，休息

4. 团队内部进行讨论，明确至少两个人对某个或几个功能重点关注（对相关业务熟悉的和不熟悉的组成一组）。我们曾经尝试大家一起对所有功能进行分析，但发现有时候没有人会提问或发言。
5. PO 向团队介绍期望的 Sprint 目标。介绍各个 Backlog 背后的用例，包括正常流程、异常流程。此时团队有任何细节问题都可以向 PO 提问以获得答案。界面原型也是讨论的一个方面。由项目经理进行整理、记录。以报表为例，此时就增加了如何排序等细节
以上持续时间：4～6 小时，
6. 团队根据项目经理的整理资料，再回顾一次 Backlog 背后的用例，有不确定的地方和 PO 讨论。同时完成各个功能的最主要验收标准
#5 持续时间：4~6 小时
7. 交付物：明确的 Sprint 周期和 Backlog 的用例资料，我们的资料包括文字描述的正常、异常流程和手工界面原型照片、最主要验收标准

有一个我们实际的例子来说明交付物的内容：（***是为了保护商业秘密）**
需求描述（User Story）：
原则：
Who, What, Why。在实践中，Why，即 Business Value，很多时候 PO 是会忽略的。
例子：
各级业务管理员可通过***平台了解全国/省/地市范围内的业务用户数、整体业务量以及***号码使用情况等信息。系统可以固定格式自动按日/周/月/年四个时间维度提供报表，报表包括：
报表 1：***业务发展情况
报表 2：***业务号码使用情况统计报表
报表 3：***号码业务流量统计报表

需求分析
原则：
保证正常流程、异常流程描述清楚即可。由于这个例子是报表，所以将报表的种类、报表中的字段定义清楚即可。
例子：
报表种类：
包括 4 类报表：关键指标（默认显示）、日报、周报、月报
查看方式：

1）系统默认显示“关键指标”，用户可以通过点击相应的按钮，切换到相应的报表页面，并显示当前最新的日/周/月报表；

2）在看到最新的报表的情况下，用户可以通过输入开始时间和结束时间，查询这个阶段内所有的日/周/月报表数据。通过这样查询出来的表，可以看到一个阶段内的趋势。

日报字段定义：

累计数：*******

到达数：*******

净增数：***** — ******。

净增数量较前日的增长率%：（***** — ****）/ ***** *100%

界面原型

原则：由开发人员和 PO 沟通用户体验，在纸面/白板上讨论，最后由美工进行加工并完成以图形方式保存

例子：******

Sprint 计划会议 2

推荐理论

过程：

1. 按照 Sprint 目标中的各项功能分出相应的任务，确保考虑到工作中所有的细节：编码、测试、代码评审、会议、学习新技术、编写文档等
2. 如果任务需时超过一天，尝试将此任务分解为几个小任务
3. 通过分解任务判断功能（Story）最初评估的工作量是多是少，如果过多评估，则和 PO 一起增加 Backlog 到 Sprint 中，否则删减或者拆分
4. 团队确认 Sprint 目标（commitment）
5. 持续时间：4～6 小时

做法：

6. 如何拆分故事

我们为什么要拆分故事：

- 故事太大，不能在一个 Sprint 内完成
- 需要对故事进行精确的评估大小

拆分故事的典型做法：

- 按照数据的边界

例如一个故事：作为超级管理员，我要看到所有的数据，因为我关心业务各个层面的发展。这个故事可以被拆分为：作为超级管理员，我要看到业务管理员的数据；作为超级管理员，我要看到客户经理的数据；作为超级管理员，我要看到集团客户的数据；

- 按照业务操作的边界

例如一个故事：客户经理可以查看客户的基本信息。如果查询条件过于复杂，而且客户信息显示逻辑也比较复杂，这个故事可以拆分为：客户经理可以按照客户创建时间搜索，显示查询到的客户数量；客户经理可以按照客户创建时间搜索，显示查询到的客户信息；

一般常用的拆分步骤为：首先按照增加、查询、编辑、删除四个环节拆分业务；对每一个环节，按照业务流程不同分支路径进行拆分（例如有 IF 条件）；对每一个路径，将路径中的节点按业务顺序分步实现；

- 按照公共模块的边界：这里的公共模块包含登录、出错处理、安全等，做法是将故事拆分为包含公共模块的故事和不包含公共模块的故事。
- 将性能等非功能性需求和功能性需求分开，先考量可以工作，再考虑做的更快。（Make it work then make it faster）

拆分故事要避免的：不要把故事拆分为任务，例如将故事拆分成编写界面、完成中间层等，这时开发人员常犯的问题，有一个办法可以防范：拆分后的子故事是否有界面、中间层、数据层中至少两个的任务，如果不是，则有可能被拆分成了任务
部分内容参考 Mike Cohn：《agile estimating and planning》

我们的实践

1. 按照 Sprint 目标中的各项功能分出相应的任务，确保考虑到工作中两个关键的细节：编码、测试。这个过程也就是设计的讨论过程，最关键的是对架构的影响、接口、数据库的影响。

2. 如果任务需时超过一天，尝试将此任务分解为几个小任务，任务的分解周期是 2～8 小时。如果有多个任务非常小，例如 Bug 修改，可以几个合在一起作为一个较大的任务。任务完成的时间是在以没有任何外界干扰为前提，例如额外的技术支持等
持续时间：8～12 小时

3. 通过分解任务判断功能（Story）最后的工作量评估。同时团队反馈本 Sprint 周期内可用的资源（MD），乘以资源利用率（需求细化的程度、写文档的时间、会议、技术支持、适当的 Buffer，例如 20%等因素），就是本 Sprint 内团队可以完成的工作量。用这个工作量去和 Sprint 目标中包含的 Backlog 的工作量比较，多删少补，以达到 PO 和团队都认可的 Sprint 目标

4. 功能已经初步落实到人（若干人）。即具体的人（若干人）对功能负责，而不是对任务负责。关键的里程碑、check point 也已经确认，目的是让团队所有人对项目的进展有总体的把握。落实的原则是：逻辑复杂的功能，多人承担；有一个经验丰富的人和业务熟悉的人并不进行满负荷承担，目的是为经验少的人提供结对帮助，同时也为可能的技术支持等留下资源。

5. 团队确认 Sprint 目标（commitment）
持续时间：2 小时

6. Scrum 使用不同阶段，如何判定 Sprint 内可以完成多少功能？
承诺驱动——Scrum 使用初期（最初的几个 Sprint）
在我们头几个 Sprint 中，由于开发小组对自己的 Velocity 没有足够的了解，对 Scrum 也没有足够的了解，所以我们用“承诺驱动”的方式来决定 Sprint 内会完成多少的任务（“故事”），以下是我们的主要步骤：

- 明确功能的优先级
- 明确此次 Sprint 要达到的目标
- 选择优先级最高的功能，分解为实现任务，并评估如何实现
- 如果 Team 承诺可以在 Sprint 内实现，则此功能加入到 Sprint Backlog 中。如果不能承诺，和 PO 商议，一个办法是拆分功能，另一个办法就是放弃这个功能
- 不断评审优先级最高的一些功能，直至 Team 不能承诺完成为止

Velocity 驱动－－Scrum 使用一段时间后

随着几个 Sprint，开发团队、管理层、PO 对于开发的效率、质量等不断调整心理预期与实际的差距，需要一个量化的指标可以正确标识团队的开发能力，同时也可以作为团队绩效考核的一个参考，这个时候就需要评定团队的 Velocity。我们是这样做的：

- 记录每个 Sprint 可以实现功能的 IMD（Ideal Man Day），实现功能的标准是任务可以发布（编码、测试、部署、文档等），而不是仅仅编码完成
- 记录每个 Sprint 实际的可以利用的资源。例如我们在第三个 Sprint 时，团队 7 个成员（A，B，C，D,E，F，G），3 个星期的开发周期，A，C，E,F,G 将全勤，B 会请假 2 天，D 最后一个星期才会加入团队，所以此次 Sprint 团队实际的可利用资源为：5*15＋13+7=95(MD)
- 计算资源利用率：实际完成功能的 IMD / 实际可利用的资源。以我们的第三个 Sprint 为例，当时完成的实际功能的 IMD 为 52IMD，资源利用率为：52 / 95 ＝ 54.7%。我总结了团队资源利用率不高的一些原因：
  a) 团队刚接触新的业务，对需求的理解花费较长的时间，同时测试也发现了实现和需求不符的地方，这是由于开发人员和 PO 的沟通较少所致
  b) 技术架构不成熟，还在不断构建，影响了实际业务层面的实现
  c) 有一些技术支持的工作
- 平均前三期 sprint 的资源利用率。我们没有按照上一期 Sprint 的资源利用率来作为参考，是因为我们对于任务的完成按照 100%完成的标准。按照前三期的平均值，更加可行
- 用第四步得到的资源利用率，乘以即将开始的 Sprint 内可以利用的资源，基本可以得到此次 Sprint 的团队开发能力（Velocity）

7. 如何做：估算任务的粒度在 2~8 小时

我们现在是用 IMD 的方式评估工作量，即每一个故事的大小，分解任务时将任务的粒度控制在 2～8 小时之内。

我们的想法：

- Sprint Planning 中分解任务部分要达到的目标，不仅仅是看到了一个计划，更重要的是如何完成计划。将任务分解为小时为单位，是为了使团队考虑如何实现功能时考虑的更加细致，更容易在团队内部讨论，及早发现问题，更加靠谱。
- 瀑布和 RUP 的开发方式和敏捷开发进行任务拆分，如果我们认为一个版本的功能要拆为 100 份工作会分析的比较透彻，那么打个比方，1000 元分为 100 份，每份 10 元；50 元分为 100 份，每份 0.5 元；瀑布开发的周期较长，整体工作量大，任务拆分的评估单位为天就不错了，而敏捷开发周期短，每个周期的功能工作量小，如果仍然以天为单位，例如 3 天，团队成员无法了解 3 天的细节，本来敏捷开发就没有特别完善的文档，那么如果在 3 天里出现了意外，团队无法进行有效的帮助。

我们是这样做的：

- 每一个故事对任务的拆分，至少包含：编码、测试、文档；如果任务超过 8 个小时，则再拆分。
- 在 Sprint 中往往有上一个 Sprint 的 Known issues，修改并不会花费很多的时间，很多小的 issue 只会花费 20 分钟，这样我们会汇总相似的，表明工作量为 2～3 个小时
- 当某一个 Story 确实无法完成时，团队可以根据任务，和 PO 讨论拆分 Story。我们有过这样一个例子：一个 Story 要求用户可以查询手机号码以获得通话记录，我们分解的任务是：查询界面、精确查询、带*、？、关联姓名等查询、查询结果界面，其中带*、？、关联姓名等查询耗时较多，最后将 Story 拆分为：用户可以精确查询手机号码以获得通话记录、用户可以模糊、关联查询手机号码以获得通话记录。其中前者在此次 Sprint 内完成

举例：

我们有一个 story：用户可以发送定时短信，以便在方便的时间编写短信，在不方便的时间发送短信。

分解的任务为：

1. 从数据库中查询满足定时发送的短信 （2H）
2. 利用第三方的发送端口发送（15H）
   2.1 编写调用第三方发送 API 的接口（5H）
   2.2 编写 API 接口的测试用例（3H）
   2.3 准备 API 接口的测试数据（2H）
   2.4 测试 API 接口（5H）
3. 显示发送状态（2H）
4. 进行界面功能的单元测试（8H）
5. 更新需求文档（1H）
6. 更新设计文档（1H）
7. 更新部署手册（1H）

（）内部分为评估的时间

一点体会：

实践中团队成员开始并不愿意以小时为单位进行拆分任务，一方面是不习惯如此细分，很琐碎，觉得是对自己的不信任，另一方面每个人都会或多或少评估工作量时给自己留一些 Buffer，以天为单位留 Buffer 容易一些，以小时为单位则任务拆分的更细，buffer 的空间就小了。

参考了《Agile Estimating and Planning》中对敏捷计划的看法：

1. 计划本身比最后的计划重要
2. 计划很容易调整
3. 如果需要，在项目过程中不断调整计划

每日例会

推荐理论

1. 参与人：团队、SM、PO（可选）、相关人员（可选）
2. 在 Sprint Backlog 上的所有任务都是可以增删修改，可重排序的任务的状态可设为“待处理”，“正在处理”，“已完成”的
3. 会议限制在 15 分钟，团队每人回答三个问题：

上次会议时的任务哪些已经完成?
——把任务从“正在处理”状态转为“已完成”状态
下一次会议之前，你计划完成什么任务？
——如果任务状态为“待处理”：转为“正在处理”状态
——如果任务不在 Sprint Backlog 上：添加这个任务
——如果任务不能在一天内完成：把这任务细分成多个任务
——如果任务可以在一天内完成：把任务状态设为“正在处理”
有什么问题阻碍了你的开发
——如果任务状态已经是“正在处理”：询问是否存在阻碍任务完成得
问题，如果有阻碍你开发进度的问题：把该障碍加入到障碍 Backlog 中

4. 如果展开了一个问题的讨论

提醒团队的成员们注意把精力集中在回答关键问题上

5. 如果相关人员想发表些言论：

礼貌地提醒他，该会议只允许让小组成员讨论

6. 交付物：障碍 Backlog、最新的 Sprint Backlog、最新的工作进展图

我们的实践

1. 参与人：团队、SM
2. 我们每天都会有这个会议（9：00 开始），一般在 20 分钟左右。每 2 天会在 40 分钟左右，这样的会议我们会每次轮换 2 人展示其最核心的一段代码（不超过 100 行），大家来 Review，包括 code standard，业务逻辑、异常处理等。

3. 每天的会议，对任务的处理都在口头上（除了可以直接移动任务），由 SM 会后整理、更新白板，否则时间会很紧张。具体的白板信息，见图（我们的白板）。说明一点：在回顾会议中大家对 3 周的很多细节不能完全回忆，会议的过程不够深入，所以决定在白板中增加一项：GBU，Good、Bad、Ugly，即在 Sprint 内无论是技术、沟通、管理各个层面，团队的任何人任何时候都可以在这一项中添加自己的意见，这样在回顾会议中，甚至在开发过程中，我们都可以反思、提高我们的流程。

4. 每天的会议，除了说这三件事，我们把它作为整个团队共享信息的机会，例如某个功能细节和 PO 确认，说给大家听；有一个技术突破，说给大家听。只是 High-Level 的层面，如果谁有兴趣，可以下来细谈

5. 所有人都会准时参加，因为公司做了考勤制度，3 次迟到了会扣除当日工资的 50% :(

6. 对于各自遇到的障碍，可以马上得到解决，则明确解决方案，否则明确到承担人，会后再想办法解决，下一次 Daily 会议时通知所有人。具体的解决障碍的方法参考图（我们这样解决障碍）

过程监控

推荐理论

下一个版本中增加。

我们的实践

1. 我们用 Scrumworks 记录谁在做什么，Sprint 的状态，和 PO 讨论产品的 Backlog。但是我们每天还是用白板做记录，见图（我们的白板）。白板对于团队和 PO 有最直接的作用，但对于管理层、其它部门、客户等需要了解 Product Backlog、Product Burndown Chart，很多时候也不会出现在团队的开发环境，Scrumworks 提供的 B/S 方式保证了异地了解的可能

2. 我们如何用 ScrumWorks 做项目管理？

在实践中，选择的工具是 Scrumworks Basic 版本（V1.8.3），各个 Stakeholders 都可以通过 Scrumworks Server 来加入讨论，了解进展。具体我们是这样做的：

- 在公共服务器上安装 Scrumworks Basic 版本，加入用户，建立权限
- PO 在“Uncommitted Backlog Items”区域中增加新的 Backlog，项目组也可以增加，由 PO 进行优先级设定。项目组对增加的 Backlog 中处于“Top”即高优先级的部分首先进行工作量评估。Backlog Item 在列表中的位置越高，表示优先级越高
- 在左侧的“Committed Backlog Items”区域中，PM 建立本次的 Sprint
- 在 Sprint Planning 之后，PM 将准备在本次 Sprint 完成的 Backlog Items 移到左侧的 Sprint 中，并建立具体的任务，并列出每一个任务评估的时间
- 每天的 Daily Meeting 后，PM/SM 会更新完成情况
- 有权限的 Stakeholders，例如管理层、客户代表、PO、项目组成员，都可以在 Sprint 的窗口内看到 Burndown chart，以了解当前的状态；同时也可以在“view impediments”窗口中了解项目当前需要的问题
- 回顾会议，“View Impediments”以及“Burndown chart”是两个重要的参考资料

说明：几乎所有的数据和记录，我们都是会议后再加入到系统中，在会议期间，白板是我们最常用的工具。

3. 我们遇到的一些项目预警迹象：

- 实际的工作量和评估的工作量不符

在我们的实践中，特别在初期，经常会经常遇到实际的工作量大大超过工作量的评估，偶尔也会遇到实际的工作量远远小于工作量的评估，这都是 Team、PO 等对自

己认识不足的原因，但有一点，我们可以从 Burn-down chart（燃尽图）中很快发现迹象。参考图（实际的工作量和评估的工作量不符）。图中红色的曲线，给 Team 明显的迹象：实际的工作量已经超过了评估的工作量，项目有极大的风险。图中蓝色的曲线，给 Team 明显的迹象：实际的工作量远远小于评估的工作量，项目没有风险，但会造成 Team 由于工作量不饱满而表现出的一些问题，特别是有大量的时间浏览与工作无关的网站等，以至于引起 PO 以及管理层对项目组的不满，从而影响双方的信任等。

- 功能不能全部完成

在 Sprint 中，如果燃尽图一直是正常的，但在某一个时间点经常会有意外的情况导致所有功能不能完全实现，此时我们的策略是：

1）按照优先级实现功能（故事）

2）功能之间是松耦合的，或者说是依赖性不强的，一定是某一个或者某一组（依赖性强）的功能完全实现后再进行下一个或者下一组。

在这个场景中，项目风险的典型迹象之一是先完成优先级低的功能，当 Sprint 结束时，高优先级的功能没有完成，提交的软件不能使 PO 和客户非常满意。这种迹象参考图（功能不能全部完成、高优先级未开发）

4. 我们的白板是怎样的？

改进前：

虽然我们用 Scrumworks 来管理 Scrum 项目，但我们仍然用白板的方式来最直接的沟通项目的状态，改进前我们是这样做的：

- 用一块（180*120 厘米）的白板作为沟通的介质。所有人从外面进入办公区，都必须经过一个小的会议室，这个白板就放在里面，我们平时会议也在这里
- 白板分为两个区域：任务进展、关键信息。任务进展区域反映有哪些功能，分解为什么任务，哪些任务正在进行，哪些任务需要确认需求，哪些功能已经达到了“DoD（definiation of done）”的标准；关键信息区域包含燃尽图（目的是使项目进度一步了然）、“新增加的功能”等 4 个信息项是为了记录 Sprint 内项目更多的细节以及在回归会议时分析项目经验教训时提供直接的信息
- 我们是每天做 Daily 会议，所以同样周期更新一次白板信息

具体的白板参考图（我们的白板－改进前）

改进后：

在我们的实践中，虽然 Scrum 的理论要求在 Sprint 内不增加功能，但实际上团队会受到销售、业务、市场，甚至管理层直接的压力，PM 或者 SM 无法拒绝新增加的功能，所以经常会出现：增加功能、移除优先级低的功能等。此时燃尽图已经无法表达 Scope 的变化，而这些变化也最容易被忽视，因为大家都在关注着最终的结果。所以在最近我们选择了“变更－燃尽柱”来反映项目的状态，同时删除了“Spike 项目”的信息，用“GBU”信息取代，原因是前者的初衷是为了表达如果此类信息太多，说明团队的技术能力需要大大提高，或者说技术方向选择的不正确，偏离了团队现有的技术能力，但发现 PO 或者管理层对这些并不关心，他们关心的是结果或者是遇到的障碍（虽然 Spike 对于团队很重要），所以我们在白板中不再标明；同时在回顾会议中大家对 3 周的很多细节不能完全回忆，会议的过程不够深入，所以决定在白板中增加一项：GBU，Good、Bad、Ugly，即在 Sprint 内无论是技术、沟通、管理各个层面，团队的任何人任何时候都可以在这一项中添加自己的意见，这样在回顾会议中，甚至在开发过程中，我们都可以反思、提高我们的流程。

我们用的 Card，是两种尺寸的“记事贴”，不记得具体的尺寸，一种的正方形的，一种是长方形的，最长的边应该在 5 厘米以内。我们首选正方形的，因为可写的内

容多一些☺

具体的白板参考图（我们的白板）

Sprint 评审会议

推荐理论

1. 参与者：PO、SM、团队
2. 团队按 Backlog 中的问题，逐个地介绍这次 Sprint 的结果，和演示新功能。
如果产品负责人想要改变功能：添加一个新问题到产品 Backlog 中
如果对功能有一个新的想法：添加一个新问题到产品 Backlog 中
如果小组报告项目遇到阻碍现在还没能解决：把该障碍加入到障碍 Backlog
3. 交付物：对这次 Sprint 的结果和整个产品的开发状态的共识

我们的实践

参与者：PO、SM、团队
过程：
在 Scrum 体系中，每一个 Sprint 之后都会做一个演示（Demo），以获得 Stakeholders 的反馈。在实践中，我们发现如果只在 Sprint 之后再做 Demo，由于 Sprint 过程中沟通不充分，Demo 展示的功能很可能不符合客户真正的需求，导致 Sprint 失败。于是，我们：
1. 将 Sprint 内的 Features（user story）按照耦合的程度分为几个组
2. 每一个组完成后会部署在开发平台上进行测试，此时的测试首先是 QA 的初步测试，保证 Feature 可以正常运行；然后是 PO 的验收测试，保证功能没有偏离需求
3. 当第二个组的 Features 完成后，会进行整合测试
4. 在 Sprint 之后，再进行整体的 Demo。
流程可以参考图（我们这样做 Demo）

好处：
1. 任何偏离需求的风险在 Sprint 期间的非正式 Demo 时被发现，得到及时处理
2. Feature 组是按照优先级排序的，先做的是最高优先级，那么如果发生了任何意外（例如人员变动、技术障碍等）无法完成所有的 Feature，那么高优先级的 Feature 可以在 Sprint 结束前被提交。

我们做 Demo，会坚持以下原则：

1. 只是演示本次 Sprint 内的功能
2. 演示过程中，Stakeholders 提出的意见，在演示会议中不做讨论，只是做记录，在会后再进行沟通。
3. 保持 Demo 时间控制在 45 分钟
4. 一个人完成所有的演示
5. 只演示已经完成的功能。除非 Stakeholders 要求，否则不展示未完成的功能。因为只有已经完成的功能才是可以给客户带来价值的。