火龙果 · 整理
uml.org.cn

硝烟中的 Scrum 和 XP

——我们如何实施 Scrum

作者：Henrik Kniberg

译者：李剑

审校：郑柯

火龙果 · 整理
uml.org.cn

目　录

译者序

致谢

序——JEFF SUTHERLAND

序——MIKE COHN

前言——嘿，SCRUM成了!

简介 1

免责声明 2
撰写本书的原因 2
SCRUM到底是什么? 3

我们怎样编写产品BACKLOG 4

额外的故事字段 6
我们如何让产品BACKLOG停留在业务层次上 7

我们怎样准备SPRINT计划 8

我们怎样制定SPRINT计划 10

为什么产品负责人必须参加 10
为什么不能在质量上让步 12
无休止的SPRINT计划会议…… 13
SPRINT 计划会议日程 14
确定SPRINT长度 15
确定SPRINT目标 16
决定SPRINT要包含的故事 17
产品负责人如何对SPRINT放哪些故事产生影响？ 18
团队怎样决定把哪些故事放到SPRINT里面？ 20

我们为何使用索引卡 26
定义“完成” 30
使用计划纸牌做时间估算 30
明确故事内容 32
把故事拆分成更小的故事 33
把故事拆分成任务 34
定下每日例会的时间地点 36
最后界限在哪里 36
技术故事 37
BUG跟踪系统 VS. 产品 BACKLOG 40
SPRINT 计划会议终于结束了！ 41

我们怎样让别人了解我们的SPRINT 42

我们怎样编写SPRINT BACKLOG 45

SPRINT BACKLOG的形式 45
任务板怎样发挥作用 47
例 1 ——首次每日SCRUM之后 47
例 2 ——几天以后 49
任务板警示标记 52
嘿，该怎样进行跟踪呢？ 55
天数估算VS. 小时估算 55

我们怎样布置团队房间 56

设计角 56

我们怎样进行每日例会 61

我们怎样更新任务板 61
处理迟到的家伙 62
处理“我不知道今天干什么”的情况 63

我们怎样进行SPRINT演示 65

火龙果 · 整理
uml.org.cn

为什么我们坚持所有的SPRINT都结束于演示 65
SPRINT演示检查列表 66
处理“无法演示”的工作 67

我们怎样做SPRINT回顾 68

为什么我们坚持所有的团队都要做回顾 68
我们如何组织回顾 69
在团队间传播经验 71
变，还是不变 71

SPRINTS之间的休整时刻 74

我们怎样制定发布计划，处理固定价格的合同 77

定义你的验收标准 77
对最重要的条目进行时间估算 78
估算生产率 80
统计一切因素，生成发布计划 81
调整发布计划 82
结对编程 83
测试驱动开发（TDD） 84
增量设计 87
代码集体所有权 88
充满信息的工作空间 88
代码标准 88
可持续的开发速度/精力充沛的工作 89

我们怎样做测试 90

你大概没法取消验收测试阶段 90
把验收测试阶段缩到最短 91
把测试人员放到SCRUM团队来提高质量 92
在每个SPRINT中少做工作来提高质量 94
验收测试应该作为SPRINT的一部分么？ 94

火龙果 · 整理
uml.org.cn

SPRINT 周期 VS. 验收测试周期 95
别把最慢的一环逼得太紧 99
回到现实 100

我们怎样管理多个SCRUM团队 101

创建多少个团队 101
为什么我们引入“团队领导”的角色 106
我们怎样在团队中分配人手 107
是否使用特定的团队？ 108
是否在SPRINT之间重新组织团队? 112
兼职团队成员 113
我们怎样进行SCRUM-OF-SCRUMS 113
交错的每日例会 116
救火团队 117
是否拆分产品BACKLOG？ 118
代码分支 123
多团队回顾 124

我们怎样管理地理位置上分布的团队 125

离岸 126
在家工作的团队成员 128

SCRUM MASTER检查列表 129

SPRINT开始阶段 129
每一天 129
在SPRINT结束时 130

额外的话 131

推荐阅读 132

有关作者 133

火龙果·整理
uml.org.cn

译者序

孙子兵法有云：兵无常势，水无常形，能因敌变化而取胜者谓之神。很多人都向往用兵如神的境界，想必也知道读万卷书不如行万里路，纸上谈兵的故事更是耳熟能详；但偏偏不能举一反三。

且看风清扬的一段话："……你将这华山派的三四十招融合贯通，设想如何一气呵成，然后全部将它忘了，忘得干干净净，一招也不可留在心中。待会便以甚么招数也没有的华山剑法，去跟田伯光对打"。如果有人说，既然"无招胜有招"是武学的最高境界，那干脆什么招数都不要学，拿把剑乱挥乱舞，处处破绽，也就处处无破绽，便是天下第一了。听到这话的人肯定会笑他太缺心眼。

我在这里不想解释为什么上面那种说法缺心眼，因为只要不是缺心眼的读者就肯定能够理解说他缺心眼的理由。但有句话叫做"不识庐山真面目，只缘身在此山中"。对待离自身尚远的事物时，人们可以把它分析的淋漓尽致；但到了自己身上，就往往成了当局者迷，旁观者清。譬如青春、譬如爱情、譬如敏捷软件开发。

我想，这本书的读者大概都知道，现如今敏捷开发是何等炙手可热的程度，但潮流一起，跟风者势必有之。虽然没法在这篇短短的序中逐一批驳，大家也可以仔细思索一下，在周边是否存在缺心眼的做法。比如，把 bad smells 背下来以后就大谈重构的好处；版本控制、缺陷跟踪、配置管理等一无所有，便一味追求持续集成；单元测试还不会写，就疯狂宣传测试驱动开发……这些都还好，只要没有把敏捷等同于迭代，等同于又敏又捷，又快又爽；这也无所谓，只要没有在实际上对敏捷一无所知、对想要达到的目标不甚了了、对项目中存在的问题视若无睹的情况下宣传敏捷、推行敏捷就可以了。但如果前面那些条件都吻合，最后这一点还能不满足么？

其实，敏捷不是说出来的，是干出来的。
是为序。

李剑

于小女出生 43 天之际

火龙果 · 整理
uml.org.cn

致谢

本书初稿完成仅用了一个周末，但很显然：那是一个超高强度工作的周末！投入程度高达 150%:o)

感谢我的妻子 Sophia 和两个孩子 Dave 与 Jenny，我那个周末扔下她们独自工作，她们对此表示了宽容；还应该感谢 Sophia 的父母——Eva 和 Jörgen，在我忙碌的时候，他们过来一起照看了整个家庭。

同时，还应该感谢在斯德哥尔摩 Crisp 工作的同事，还有 scrumdevelopment yahoo 讨论组的成员，他们一起校稿，提出了很多改进意见。

最后，我要深深感谢所有的读者，从你们长期的反馈中我收获颇丰。尤其要指出，能够通过本书点燃许多人尝试敏捷软件开发的热情，这让我感到特别开心！

序——Jeff Sutherland

开发团队需要了解一些 Scrum 的基础知识。该怎样创建产品 Backlog，对它进行估算？怎样把它转化成 Sprint Backlog？怎样管理燃尽图（Burndown chart），计算团队的生产率（velocity）？Henrik 的书可以用作一些基础实践的入门指南，帮助团队从试用 Scrum 中成长，最终成功地实施 Scrum。

当前，良好的 Scrum 执行过程对需要风险投资的团队正变得日益重要。我现在是一个风险投资团队的敏捷教练；为了帮助他们达成目标，我给出的建议是：只给敏捷实践实施情况良好的敏捷公司投资。团队中的资深合伙人（Senior Partner）在向所有的待投资企业问同一个问题：你们是否清楚团队的生产率？目前他们都很难做出明确的答复。要想在将来得到投资，开发团队就必须清楚自己的软件生产率。

为什么这一点如此重要呢？如果团队不清楚自己的生产率，那么产品负责人（产品负责人）就无法用可靠的发布日期来创建产品路线图。如果没有可靠的发布日期，公司的产品就可能会失败，投资人的钱就有可能化为乌有！

无论公司规模大小，创办时间长短，或者是否有资金注入，这个问题都是它们所要面对的。在最近在伦敦举办的一个大会上，我们曾讨论过 Google 内部的 Scrum 实施状况，当时的听众有 135 个人，我问他们中有多少人在使用 Scrum，只有 30 个举手。我又接着问他们是否在根据 Nokia 标准来做迭代开发。迭代开发是敏捷宣言的基本原则——在早期频繁的交付可工作的软件。Nokia 用了几年时间，对上百个 Scrum 团队的工作进行了回顾，总结出了迭代开发的基本需求：

- 迭代要有固定时长（被称为“时间盒——timebox”），不能超过六个星期。
- 在每一次迭代的结尾，代码都必须经过 QA 的测试，能够正常工作。

使用 Scrum 的 30 个人里面，只有一半人说他们遵守了 Nokia 标准，符合敏捷宣言的首要原则。我又问他们是否遵守了 Nokia 的 Scrum 标准：

- Scrum 团队必须要有产品负责人，而且团队都清楚这个人是谁。
- 产品负责人必须要有产品 Backlog，其中包括团队对它进行的估算。
- 团队必须要有燃尽图，而且要了解他们自己的生产率。
- 在一个 Sprint 中，外人不能干涉团队的工作。

仅有的 30 个实践 Scrum 的在场人士中，只有 3 个能够通过 Nokia 的 Scrum 测试。看来只有这几个团队才有可能在将来得到我的风投伙伴的钱了。

如果按照 Henrik 列出的实践执行，那么你就会拥有如下产物：产品 Backlog、对于这个 Backlog 的估算、燃尽图；你会了解团队的生产率，并掌握在切实有效的 Scrum 过程中所包含的众多基础实践。这些收获就是本书的价值所在。你将会通过 Nokia 的 Scrum 测试，对工作的投资也会物有所值。如果你的公司还正处于创业阶段，也许还会收到风险投资团队的资金注入。你也许会塑造软件开发的未来，成为下一代软件产品中领军产品的创建者。

Jeff Sutherland,
Ph.D., Co-Creator of Scrum

序——Mike Cohn

Scrum 和极限编程（XP）都要求团队在每一次迭代的结尾完成一些可以交付的工作片段。迭代要短，有时间限制。将注意力集中于在短时间内交付可工作的代码，这就意味着 Scrum 和 XP 团队没有时间进行理论研究。他们不会花时间用建模工具来画 UML 图、编写完美的需求文档，也不会为了应对在可预计的未来中所有可能发生的变化而去写代码。实际上，Scrum 和 XP 都关注如何把事情做好。这些团队承认在开发过程中会犯错，但是他们明白：要投入实践中，动手去构建产品，这才是找出错误的最好方式；不要只是停留在理论层次上对软件进行分析和设计。

注重实践而非理论研究，这正是本书的独到之处。Henrik Kniberg 很清楚，初涉门径的人更需要这种书籍。他没有对“什么是 Scrum”进行冗长的描述，只是给出了一些网站作为参考。从一开始他就在讲他的团队如何管理产品 Backlog，并基于它进行工作。接着他又讲述了成功的敏捷项目中包含的所有元素和实践。没有理论、没有引用、没有脚注、没有废话。Henrik 的书没有从哲学角度上分析为什么 Scrum 可以工作，没有分析为什么你可能会尝试不同的选择。它描述的是一个成功敏捷团队的工作过程。

所以本书的副标题——“我们如何实施 Scrum”——才显得格外贴切。这也许不是**你**实施 Scrum 的方式，这是 Henrik 的团队实施 Scrum 的方式。你也许会问：“为什么我需要关心别的团队怎样实施 Scrum？”这是因为通过关注其他团队的实施过程，尤其是成功的案例，我们就可以学到更好的实施方式。这不是，也永远不会是“Scrum 最佳实践”的罗列，因为团队和项目的真实场景要比其他一切都重要的多。我们应该了解的是优秀实践及其应用范围，而不是最佳实践。在读过足够多的成功团队的实践经验以后，你便会做好充分的准备，来面对实施 Scrum 和 XP 的过程中将会遇到的艰难险阻。

火龙果 · 整理
uml.org.cn

Henrik 提供了很多优秀实践，还有对应的使用场景；通过它们，我们能够更好地掌握如何在自己的项目中，在战壕里使用 Scrum 和 XP。

Mike Cohn
***Agile Estimating and Planning* 和 *User Stories Applied for Agile Software Development* 的作者**

前言——嘿，Scrum 成了!

Scrum 成了！至少对我们来说它已经成功了（这里指的是我当前在 Stockholm 的客户，名字略过不提）。希望它对你们也一样有用！也许这本书会对你们实施 Scrum 的过程有所助益。

这是我第一次看到一种开发方法论（哦，对不起，Ken，它是一种**框架**）可以脱离开书本成功运作。它拿来就能用。所有人——包括开发人员、测试人员和经理——都为此而高兴。它帮助我们走出了艰难的境地，而且让我们在严重的市场动荡和大规模的公司裁员中依然能够集中精力在项目上。

我不该说我为此感到惊讶，但实情确实如此。在一开始我大致的翻了几本讲 Scrum 的书，它们把 Scrum 描述的挺不错，却给我留下了一种太过美好以致不太真实的感觉（我们都知道“某些东西看上去太好了……”这类说法的含义）。所以我没法不对它有点怀疑。但在使用 Scrum 一年以后，先前的零星疑虑早已烟消云散。我被它深深的震撼了（我们团队中的大部分人都和我一样），以后只要没有充分的理由来阻止我，我都会继续使用 Scrum。

火龙果 · 整理
uml.org.cn

1

简介

你即将在组织中开始使用 Scrum。或者你已经用过了几个月。你已经了解了基础概念，读过了几本书，也许你还已经通过了 Scrum Master 认证。先恭喜一下！

但是你仍会感到迷茫。

用Ken Schwaber的话说，Scrum不是方法学，它是一个**框架**。也就是说Scrum不会告诉你到底该做些什么。靠！

下面有一个好消息和一个坏消息。好消息是我即将和你们分享我使用 Scrum 的经验，还有种种恼人的细节。而坏消息是，这只是我个人的经历。**你**不应该完全仿照我的做法。实际上如果换个不同的场景，我也许就会换种实践方式了。

Scrum 的强大和令人痛苦之处就在于你不得不根据自己的具体情况来对它进行调整。

过去的一年中，我在一个大约 40 人的开发团队里面试验性地使用了 Scrum。当时公司正处于困境，没日没夜地加班，产品质量低下，很多人都忙着四处救火，交付日期也一再拖延。公司已经决定了使用 Scrum，但是并没有完全落实，剩下的部分就是我的工作了。在那个时候，对团队中的大多数人来说，“Scrum”就只是一个陌生的、时时能够从走廊上听到回音的时髦词汇；如此而已，和他们日常的工作没有丝毫的关系。

火龙果 · 整理
uml.org.cn

一年过去了，我们在公司里从上到下都实现了 Scrum。我们试过多种团队尺寸（3-12 人）、sprint 长度（2-6 个星期）；定义“完成”的不同方式；不同形式的产品 backlog 和 sprint backlog（Excel、Jira、索引卡）；多种测试策略、演示方式、多个 Scrum 团队的信息同步方式……。我们还试验了 XP 实践——各种各样的每日构建，结对编程，测试驱动开发，等等；还试过把 XP 和 Scrum 进行结合。

这是一个持续学习的过程，所以故事尚未结束。我相信，如果公司能够保持做 sprint 回顾的良好习惯，他们就会不断得到新的收获，重新领悟到怎样在他们特有的场景中，把 Scrum 用得恰到好处。

免责声明

这篇文档讲述的不是“正确”实现 Scrum 的方式！它只是表明了一种方式，是我们在一年内不断修正调整后的结果。你也可以认为我们的做法是完全错误的。

本文中所说的一切都是我个人的观点，不代表 Crisp 或者我当前客户的任何意见。因此我将避免提到任何特定的产品或者人名。

撰写本书的原因

在学习 Scrum 的过程中，我读过了 Scrum 和敏捷方面的书，浏览了许多有关 Scrum 的网站和论坛，通过了 Ken Schwaber 的认证，用各种问题刁难他，还花了大量的时间跟同事进行讨论。但在纷乱芜杂的信息中，我感到最有价值的就是那些真枪实弹的故事。它们把“原则与实践”变成了……嗯……“如何真正动手去做的过程”，同时还帮我意识到（有时候会帮我避免）Scrum 新丁容易犯的典型错误。

所以，现在轮到我做出一些回报了。下面就是我以 Scrum 为枪的战斗经历。

希望本书对有同样经历的读者起到抛砖引玉的作用，给我反馈，向我开炮！

Scrum 到底是什么?

哦，对不起，你完全不了解 Scrum 或者 XP？那你最好先去看一下这几个链接：

- http://agilemanifesto.org/
- http://www.mountaingoatsoftware.com/scrum
- http://www.xprogramming.com/xpmag/whatisxp.htm

要是你真没耐心去访问这些网站，也没关系。随便翻翻看看吧。大多数 Scrum 的相关术语都会在书中慢慢讲到，你会感兴趣的。

火龙果 · 整理
uml.org.cn

2

我们怎样编写产品 backlog

产品 backlog 是 Scrum 的核心，也是一切的起源。从根本上说，它就是一个需求、或故事、或特性等组成的列表，按照重要性的级别进行了排序。它里面包含的是客户想要的东西，并用客户的术语加以描述。

我们叫它**故事**（*story*），有时候也叫做 *backlog* **条目**。

我们的故事包括这样一些字段：

- **ID**——统一标识符，就是个自增长的数字而已。以防重命名故事以后找不到它们。
- **Name（名称）**——简短的、描述性的故事名。比如“查看你自己的交易明细”。它必须要含义明确，这样开发人员和产品负责人才能大致明白我们说的是什么东西，跟其他故事区分开。它一般由 2 到 10 个字组成。
- **Importance（重要性）**——产品负责人评出一个数值，指示这个故事有多重要。例如 10 或 150。分数越高越重要。
    - 我一直都想避免“优先级”这个说法，因为一般说来优先级 1 都表示“最高”优先级，如果后来有其他更重要的东西就麻烦了。它的优先级评级应该是什么呢？优先级 0？优先级-1？
- **Initial estimate（初始估算）**——团队的初步估算，表示与其他故事相比，完成该故事所需的工作量。最小的单位是

火龙果 · 整理
uml.org.cn

故事点（story point），一般大致相当于一个“理想的人天（man-day）”。

- 问一下你的团队，“如果可以投入最适合的人员来完成这个故事（人数要适中，通常为 2 个），把你们锁到一个屋子里，有很多食物，在完全没有打扰的情况下工作，那么需要几天，才能给出一个经过测试验证，可以交付的完整实现呢？”如果答案是“把 3 个人关在一起，大约需要 4 天时间”，那么初始估算的结果就是 12 个故事点。
- 不需要保证这个估值绝对无误（比如两个故事点的故事就应该花两天时间），而是要保证相对的正确性（即，两个点的故事所花费的时间应该是四个点的故事所需的一半）

- **How to demo（如何做演示）**——它大略描述了这个故事应该如何在 sprint 演示上进行示范，本质就是一个简单的测试规范。“先这样做，然后那样做，就应该得到……的结果 ”。
  - 如果你在使用 TDD（测试驱动开发），那么这段描述就可以作为验收测试的伪码表示。
- **Notes（注解）**——相关信息、解释说明和对其它资料的引用等等。一般都非常简短。

产品 BACKLOG（示例）					
ID	**Name**	**Imp**	**Est**	**How to demo**	**Notes**
1	存款	30	5	登录，打开存款界面，存入 10 欧元，转到我的账户余额界面，检查我的余额增加了 10 欧元。	需要 UML 顺序图。目前不需要考虑加密的问题。
2	查看自己的交易明细	10	8	登录，点击“交易”，存入一笔款项。返回交易页面，看到新的存款显示在页面上。	使用分页技术避免大规模的数据库查询。和查看用户列表的设计相似。

火龙果 · 整理
uml.org.cn

我们曾试过很多字段，但最后发现，只有上面提到的六个字段我们会一直使用下去。

通常我们会把 backlog 存放在共享的 Excel 文档里面（是为了多个用户可以同时编辑它）。虽然正规意义上这个文档应该归产品负责人所有，但是我们并不想把其他用户排斥在外。开发人员常常要打开这个文档，弄清一些事情，或者修改估算值。

基于同样原因，我们没有把这个文档放到版本控制仓库上，而是放到共享的驱动器里面。我们发现，要想保证多用户同时编辑而不会导致锁操作或是合并冲突，这是最简单的方式。

但是基本上其它所有的制品都放在了版本控制仓库中。

额外的故事字段

有时为了便于产品负责人判断优先级别，我们也会在产品 backlog 中使用一些其它字段。

- **Track（类别）**——当前故事的大致分类，例如“后台系统”或“优化”。这样产品负责人就可以很容易选出所有的“优化”条目，把它们的级别都设得比较低。类似的操作执行起来都很方便。
- **Components（组件）**——通常在 Excel 文档中用“复选框”实现，例如“数据库，服务器，客户端”。团队或者产品负责人可以在这里进行标识，以明确哪些技术组件在这个故事的实现中会被包含进来。这种做法在多个 Scrum 团队协作的时候很有用——比如一个后台系统团队和一个客户端团队——他们很容易知道自己应当对哪些故事负责。
- **Requestor（请求者）**——产品负责人可能需要记录是哪个客户或相关干系人最先提出了这项需求，在后续开发过程中向他提供反馈。
- **Bug tracking ID（Bug 跟踪 ID）**——如果你有个 bug 跟踪系统，就像我们用的 Jira 一样，那么了解一下故事与 bug 之间的直接联系就会对你很有帮助。

火龙果 · 整理
uml.org.cn

我们如何让产品 backlog 停留在业务层次上

如果产品负责人有技术相关的背景，那他就可能添加这样一个故事：“给 Events 表添加索引”。他为啥要这么做？真正的潜在目标也许是“要提高在后台系统中搜索事件表单的响应速度”。

到后面我们可能会发现：索引并不是带来表单速度变慢的瓶颈。也许原因与索引完全不相干。指出**如何**解决问题的应该是开发团队，产品负责人只需要关注业务目标。

只要发现这种面向技术的故事，我一般都会问产品负责人 “但是**为什么呢**”这样的问题，一直问下去，直到我们发现内在的目标为止。然后再用真正的目标来改写这个故事（“提高在后台系统中搜索并生成表单的响应速度”）。最开始的技术描述只会作为一个注解存在（“为事件表添加索引可能会解决这个问题”）。

火龙果 · 整理
uml.org.cn

3

我们怎样准备 sprint 计划

Sprint 计划的这一天很快就要到来了。有个教训我们一再体会：

在 sprint 计划会议之前，要确保产品 backlog 的井然有序。

但这到底是什么意思？所有的故事都必须定义得完美无缺？所有的估算都必须正确无误？所有的先后次序都必须固定不变？不，不，绝不是这样！它表示的意思是：

- 产品 backlog 必须存在（你能想象到这一点么？）。
- 只能有一个产品 backlog 和一个产品负责人（对于一个产品而言）。
- 所有重要的 backlog 条目都已经根据重要性被评过分，不同的重要程度对应不同的分数。
    - 其实，重要程度比较低的 backlog 条目，评分相同也没关系，因为它们在这次 sprint 计划会议上可能根本不会被提出来。
    - 无论任何故事，只要产品负责人相信它会在下一个 sprint 实现，那它就应该被划分到一个特有的重要性层次。
    - 分数只是用来根据重要性对 backlog 条目排序。假如 A 的分数是 20，而 B 的分数是 100，那仅仅是说明 B 比 A 重要而已，绝不意味着 B 比 A 重要五倍。如果 B 的分数是 21 而不是 100，含义也是一样的！

- 最好在分数之间留出适当间隔，以防后面出现一个 C，比 A 重要而不如 B 重要。当然我们也可以给 C 打一个 20.5 分，但这样看上去就很难看了，所以我们还是留出间隔来！

- 产品负责人应当**理解**每个故事的含义（通常故事都是由他来编写的，但是有的时候其他人也会添加一些请求，产品负责人对它们划分先后次序）。他不需要知道每个故事具体是如何实现的，但是他要知道为什么这个故事会在这里。

注意：产品负责人之外的人也可以向产品 backlog 中添加故事，但是他们不能说这个故事有多重要，这是产品负责人独有的权利。他们也不能添加时间估算，这是开发团队独有的权利。

我们还曾经尝试过、或者评估过其它方式：

- 使用 Jira（我们的 bug 跟踪系统）存放产品 backlog。但是大多数产品负责人都觉得这东西操作起来太繁琐了。Excel 操作起来简单方便，直截了当。你可以使用不同的颜色、重新组织条目、在特定情况下添加列、添加注解和导入导出数据等等。
- 使用 VersionOne、ScrumWorks、XPlanner 这种敏捷过程工具。我们还没有测试过它们，不过以后可能会吧。

火龙果 · 整理
uml.org.cn

4

我们怎样制定 sprint 计划

Sprint 计划会议非常关键，应该算是 Scrum 中最重要的活动（这当然是我的主观意见）。要是它执行的不好，整个 sprint 甚至都会被毁掉。

举办 Sprint 计划会议，是为了让团队获得足够的信息，能够在几个星期内不受干扰地工作，也是为了让产品负责人能对此有充分的信心。

OK，这么说可能比较模糊。其实，Sprint 计划会议会产生一些实实在在的成果：

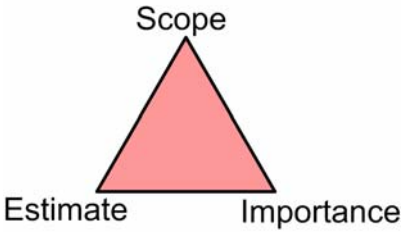
- sprint 目标。
- 团队成员名单（以及他们的投入程度，如果不是 100%的话）。
- sprint backlog（即 sprint 中包括的故事列表）。
- 确定好 sprint 演示日期。
- 确定好时间地点，供举行每日 scrum 会议。

为什么产品负责人必须参加

有时候产品负责人会不太情愿跟团队一起花上几个小时制定 sprint 计划。“嘿，小伙子们，我想要的东西已经列下来了，我没时间参加你们的计划会议。”这可是个非常严重的问题。

火龙果 · 整理
uml.org.cn

为什么整个团队和产品负责人都必须要参加 sprint 计划会议？原因在于，每个故事都含有三个变量，它们两两之间都对彼此有着强烈依赖。



范围（scope）和重要性（importance）由产品负责人设置。估算（estimate）由团队设置。在 sprint 计划会议上，经过团队和产品负责人面对面的对话，这三个变量会逐步得到调整优化。

会议启动以后，产品负责人一般会先概括一下希望在这个 sprint 中达成的目标，还有他认为最重要的故事。接下来，团队从最重要的故事开始逐一讨论每个故事，一一估算时间。在这个过程中，他们会针对范围提出些重要问题：“‘删除用户’这个故事，需不需要遍历这个用户所有尚未执行的事务，把它们统统取消？”有时答复会让他们感到惊讶，促使他们调整估算。

在某些情况下，团队对故事做出的时间估算，跟产品负责人的想法不太一样。这可能会让他调整故事的重要性；或者修改故事的范围，导致团队重新估算，然后一连串诸如此类的后续反应。

这种直接的协作形式是 Scrum 的基础，也是所有敏捷软件开发的基础。

如果产品负责人还是坚持没时间参加怎么办？一般我会按顺序尝试下面的策略：

- 试着让产品负责人理解，为什么他的直接参与事关项目成败，希望他可以改变念头。

火龙果 · 整理
uml.org.cn

- 试着在团队中找到某个人，让他在会议中充当产品负责人的代表。告诉产品负责人，“既然你没法来开会，我们这次会让 Jeff 代表你参加。他会替你在会议中行使权利，改变故事的优先级和范围。我建议，你最好在会议开始前尽可能跟他沟通到位。如果你不喜欢 Jeff 当代表，也可以推荐其他人，只要他能全程参加我们的会议就行。”
- 试着说服管理团队为我们分配新的产品负责人。
- 推迟 sprint 的启动日期，直到产品负责人找到时间参会为止。同时拒绝承诺任何交付。让团队每天都可以自由做他们最想做的事情。

为什么不能在质量上让步

在上面的三角形中，我有意忽略了第四个变量——**质量**。

我尽力把**内部质量**和**外部质量**分开。

- **外部质量**是系统用户可以感知的。运行缓慢、让人迷糊的用户界面就属于外部质量低劣。
- **内部质量一般**指用户看不到的要素，它们对系统的可维护性有深远影响。可维护性包括系统设计的一致性、测试覆盖率、代码可读性和重构等等。

一般来说，系统内部质量优秀，外部质量仍有可能很差。而内部质量差的系统，外部质量肯定也不怎么样。松散的沙滩上怎么可能建起精美的楼阁？

我把外部质量也看作范围的一部分。有时出于业务考虑，可能会先发布一个系统版本，其中用户界面给人的感觉可能比较简陋，而且反应也很慢；不过随后会发布一个干净的版本。我都是让产品负责人做权衡，因为他是负责定义项目范围的人。

不过内部质量就没什么好说的了。不管什么时候，团队都要保证系统质量，这一点毋庸置疑，也没有折扣可讲。现在如此、将来如此、一直如此，直到永远。

火龙果·整理
uml.org.cn

（嗯，好吧，**差不多**直到永远）

那么我们怎样区分哪些问题属于内部质量，哪些属于外部质量呢？假设产品负责人这样说，“好吧，你们把它估算成 6 个故事点也行。但我相信：一定能够找到些临时方案，节省一半时间。你们只要稍稍动下脑子就行。”

啊哈！他想把内部质量当作变量来处理。我是怎么知道的？因为他想让我们缩减故事的估算时间，但不想为缩减范围“买单”。“临时方案”这个词应当在你脑中敲响警钟……

为什么不允许这样干？

经验告诉我：牺牲内部质量是一个糟糕透顶的想法。现在节省下来一点时间，接下来的日子里你就要一直为它付出代价。一旦我们放松要求，允许代码库中暗藏问题，后面就很难恢复质量了。

碰到这种状况，我就会试着把话题转回到范围上来。“既然你想尽早得到这个特性，那我们能不能把范围缩小一点？这样实现时间就能缩短。也许我们可以简化错误处理的功能，把“高级错误处理”当作一个单独的故事，放到以后再实现。或者也可以降低其他故事的优先级，好让我们集中处理这一个。”

一旦产品负责人弄清楚内部质量是不可能让步的，他一般都会处理好其他变量。

无休止的 sprint 计划会议……

在 sprint 计划会议中最困难的事情是：

1） 人们认为他们花不了多长时间

2） ……但他们会的！

Scrum 中的一切事情都有时间盒。我喜欢这条简单如一的规则，并一直力求贯彻到底。

火龙果 · 整理
uml.org.cn

假如 sprint 计划会议接近尾声，但仍然没有得出 sprint 目标或者 sprint backlog，这时该怎么办？我们要打断它么？还是再延期一个小时？或者到时间就结束会议，然后明天继续？
这种事情会一再发生，尤其是在新团队身上。你会怎么做？我不知道。但我们的做法是什么？嗯……我通常会直接打断会议，中止它，让这个 sprint 给大家点儿罪受吧。具体一点，我会告诉团队和产品负责人：“这个会议要在 10 分钟以后结束。我们到目前为止还没有一个真正的 sprint 计划。是按照已经得出的结论去执行，还是明早 8 点再开个 4 小时的会？”你可以猜一下他们会怎么回答……:o)

我也试过让会议延续下去。但一般都没啥效果，因为大家都很累了。如果他们在 2 到 8 个小时（不管多久，只要你固定好时间长度就可以）内都没整理出一个还说得过去的 sprint 计划，再来一个小时他们仍然得不出结论。我们也可以明天再安排一次会议——但大家都已经耐心耗尽，只想启动这个 sprint，不想再花一个小时做计划。如果可以罔顾这个事实，那这个选择也确实不错。

所以我会打断会议。是的，这个 sprint 让大家不太好过。但我们应该看到它的正面影响，整个团队都从中获益匪浅，下个 sprint 计划会议会更有效率。另外，如果他们从前还觉得你定下的会议时间过长的话，下次他们的抵制情绪就会少一些了。

学会按照时间盒安排工作，学会制定合乎情理的时间盒，这对会议长度和 sprint 长度同样有帮助。

Sprint 计划会议日程

在 sprint 计划会议之前先为它初步制定一个时间表，可以减少打破时间盒的风险。

下面来看一下我们用到的一个典型的时间表。

Sprint 计划会议：13:00 – 17:00 （每小时休息 10 分钟）

火龙果 · 整理
uml.org.cn

- **13:00 – 13:30**。产品负责人对 sprint 目标进行总体介绍，概括产品 backlog。定下演示的时间地点。
- **13:30 – 15:00**。团队估算时间，在必要的情况下拆分 backlog 条目。产品负责人在必要时修改重要性评分。理清每个条目的含义。所有重要性高的 backlog 条目都要填写“如何演示”。
- **15:00 – 16:00**。团队选择要放入 sprint 中的故事。计算生产率，用作核查工作安排的基础。
- **16:00 – 17:00**。为每日 scrum 会议（以下简称每日例会）安排固定的时间地点（如果和上次不同的话）。把故事进一步拆分成任务。

这个日程绝不是强制执行的。Scrum master 根据会议进程的需要，可以对各个阶段的子进程时间安排进行调整。

确定 sprint 长度

Sprint 演示日期是 sprint 计划会议的产出物，它被确定下来以后，也就确定了 sprint 的长度。

那 sprint 应该多长才好？

嗯，时间短就好。公司会因此而变得“敏捷”，有利于随机应变。短的 sprint=短反馈周期=更频繁的交付=更频繁的客户反馈=在错误方向上花的时间更少=学习和改进的速度更快，众多好处接踵而来。

但是，时间长的 sprint 也不错。团队可以有更多时间充分准备、解决发生的问题、继续达成 sprint 目标，你也不会被接二连三的 sprint 计划会议、演示等等压得不堪重负。

产品负责人一般会喜欢短一点的sprint，而开发人员喜欢时间长的sprint。所以sprint的长度是妥协后的产物。做过多次实验后，我们最终总结出了最喜欢的长度：三个星期。绝大部分团队的sprint长

火龙果 · 整理
uml.org.cn

度都是三周。它不长不短，既让我们拥有足够的敏捷性，又让团队进入“流”[1]的状态，同时还可以解决sprint中出现的问题。

此外我们还发现：刚开始**要**试验 sprint 的长度。不要浪费太多时间做**分析**。选一个可以接受的长度先开始再说，等做完一两个 sprint 再进行调整。

不过，确定了自己最喜欢的长度之后，就要在长时间内**坚持住**。经过几个月的实验后，我们发现 3 个星期是个不错的长度，于是我们就把 sprint 固定为 3 个星期，进行了一段时间。有的时候会稍稍感觉有点长，有的时候感觉有点短。但保持住这个长度以后，它似乎变成了大家共同的心跳节奏，每个人都感觉很舒服。这段时间内也无须讨论发布日期之类的事情，因为大家都知道：每过三周都会有一个发布。

确定 sprint 目标

几乎每次 sprint 计划会议都要确定 sprint 目标。在 sprint 计划会议进行中，我会选某个时刻问一个问题，“这个 sprint 的目标是什么？”每个人都目光空洞的看着我，产品负责人也皱起眉头，开始挠下巴。

出于某些原因，制定 sprint 目标确实**很困难**。但我发现即使是像挤牙膏一样把它挤出来，那也是值得的。半死不活的目标也比啥都没有强。这个目标可以是“挣更多的钱”，或者“完成优先级排到最前面的三个故事”，或“打动 CEO”，或“把系统做的足够好，可以作为 beta 版发布给真正的用户使用”，或“添加基本的后台系统支持”等等。它必须用业务术语表达，而不是技术词汇，让团队以外的人也能够理解。

---

[1] 译者注：心理学家米哈里齐克森·米哈里 (Mihaly Csikszentmihalyi) 将流 (flow) 定义为一种将个人精神力完全投注在某种活动上的感觉；流产生时同时会有高度的兴奋及充实感。

火龙果 · 整理
uml.org.cn

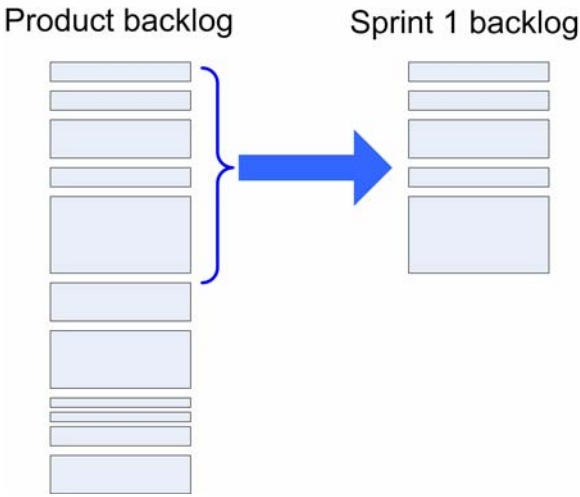
Sprint 目标需要回答这个根本的问题，“我们**为什么**要进行这个 sprint？为什么我们不直接放假算了？”要想从产品负责人的口中哄骗出 sprint 目标，你不妨一字不差的问他这个问题看看。

Sprint 目标应该是尚未达成的。“打动 CEO”这个目标不错，可如果这个系统已经给他留下了深刻印象，那就算了。这种状况下大家都可以放假回家，sprint 目标依然能完成。

制定 sprint 计划的时候，这个目标可能看上去既愚蠢又勉强，但它在 sprint 中常常会被用到，到那时大家就会开始对他们应该做啥感到困惑。如果有多个 Scrum 团队（像我们一样）开发不同产品，你可以在一个 wiki 页面(或其他东西)上列出所有团队的 sprint 目标，然后把它们放到一个显著位置上，保证公司所有人（不只是顶级管理层）知道公司在干什么，目的又是什么！

决定 sprint 要包含的故事

决定哪些故事需要在这个 sprint 中完成，是 sprint 计划会议的一个主要活动。更具体地说，就是哪些故事需要从产品 backlog 拷贝到 sprint backlog 中。



火龙果 · 整理
uml.org.cn

看一下上面这幅图。每个矩形都表示一个故事，按重要性排序。最重要的故事在列表顶部。矩形尺寸表示故事大小（也就是以故事点估算的时间长短）。蓝括号的高度表示团队的**估算生产率**，也即团队认为他们在下一个 sprint 中所能完成的故事点数。

右侧的 sprint backlog 是产品 backlog 中的一个故事快照。它表示团队在这个 sprint 中承诺要完成的故事。

在 sprint 中包含多少故事由**团队**决定，而不是产品负责人或其他人。

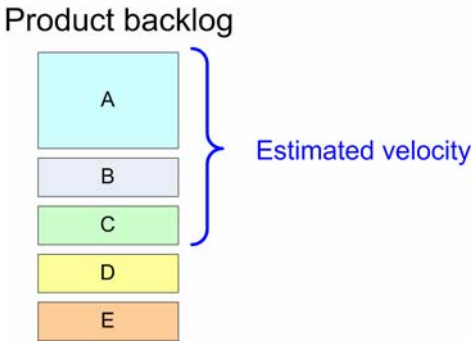
这便引起了两个问题：

1. 团队怎么决定把哪些故事放到 sprint 里面？
2. 产品负责人怎么影响他们的决定？

我会先回答第二个问题。

产品负责人如何对 sprint 放哪些故事产生影响？

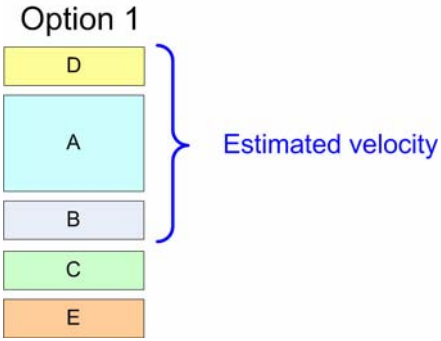
假设在 sprint 计划会议中我们遇到下面的情况。



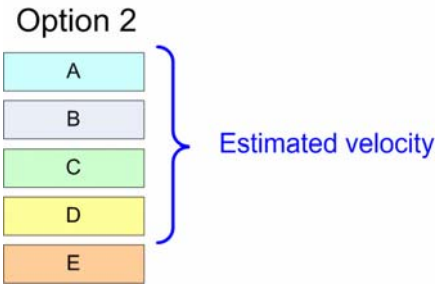
产品负责人很失望，因为故事 D 不会被放到 sprint 里面。那他在 sprint 计划会议上能做些什么?

火龙果 · 整理
uml.org.cn

首先，他可以重新设置优先级。如果他给故事 D 赋予最高的重要级别，团队就不得不把它先放到 sprint 里面来（在这里需要把 C 扔出去）。

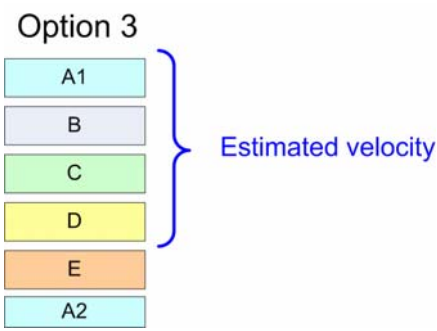


其次，他可以更改范围——缩小故事 A 的范围，直到团队相信故事 D 能在这个 sprint 里完成为止。



最后，他还可以拆分故事。产品负责人判断出故事 A 中某些方面实际并不重要，所以他把 A 分成两个故事 A1 和 A2，赋给它们不同的重要级别。

火龙果 · 整理
uml.org.cn



你可以看到，虽然产品负责人在正常情况下不能控制团队的估算生产率，他依然有很多种方式，对 sprint 中放入哪些故事施加影响。

团队怎样决定把哪些故事放到 sprint 里面？

我们在这里使用两个技术：

1. 本能反应
2. 生产率计算

用本能反应来估算

- **Scrum master**：“伙计们，我们在这个 sprint 里面能完成故事 A 吗？”（指向产品 backlog 中最重要的条目）
- **Lisa**：“呃。当然可以。我们有三个星期，这只是个微不足道的特性。
- **Scrum master**：“OK，那加上 B 怎么样？”（指向第二重要的条目）
- **Tom 和 Lisa 一起回答：“自然没问题。”**
- **Scrum master**：“OK，那 A、B、C 一起呢？”
- **Sam （对产品负责人说）**：“故事 C 要包括高级错误处理么？”
- **产品负责人**：“不，你现在可以跳过它，只需要完成基本的错误处理。”
- **Sam**：“那 C 应该没问题。”
- **Scrum master**：“OK，那再加上 D 呢？”
- **Lisa**：“嗯……”

- **Tom**：“我觉得能完成。”
- **Scrum master**：“有多少把握？90%？还是 50%？”
- **Lisa 和 Tom**：“差不多 90%”
- **Scrum master**：“OK，D 也加进来。那再加上 E 呢？”
- **Sam**：“也许吧。”
- **Scrum master**：“90%? 50%?”
- **Sam**：“差不多 50%”
- **Lisa**：“我没把握。”
- **Scrum master**：“OK，那先把它放一边去。我们要做完 A、B、C 和 D。如果有时间的话当然还可以做完 E，不过既然没人指望它能做完，所以我们不会把它算到计划里面来。现在怎么样？”
- **所有人**：“OK！”

如果 sprint 时间不长，小团队根据直觉进行估算可以收到很好的效果

用生产率计算来估算

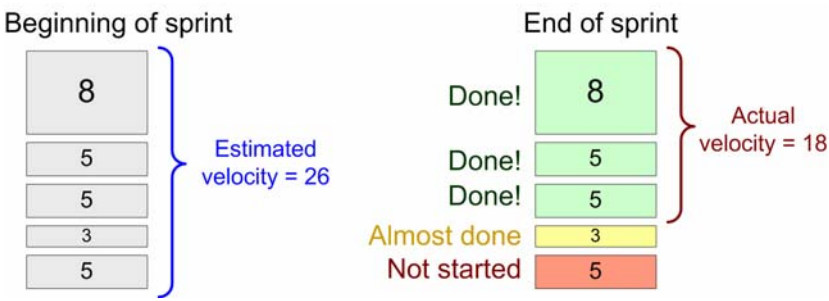
这项技术包括两步：

1. 得出**估算生产率**
2. 计算在不超出估算生产率的情况下可以加入多少故事。

生产率是“已完成工作总量”的一个衡量方式，其中每一个条目都是用它的原始估算进行衡量的。

下图中，左边是 sprint 启动时的**估算生产率**，右边是 sprint 结束时的**实际生产率**。每个矩形都是一个故事，里面的数字表示这个故事的原始估算。

火龙果 · 整理
uml.org.cn



注意，这里的实际生产率建立在每个故事的**原始**估算基础之上。在 sprint 过程中对故事时间进行的修改都被忽略了。

我已经能听到你的抱怨了："那不是闲的蛋疼么？你丫想想，得有多少事影响生产率啊？有那么一群傻拉吧唧的程序员、原始估算能错到姥姥家去、范围变化了连个响都听不到，还有，鬼知道从哪个旮旯里就能出来个东西影响我们，这种事不是太多了么！"

我同意，这个数字并不精确。但它依然很有用，尤其是与啥都没有相比，感觉就更明显了。它可以给你一些硬生生的事实："抛开具体原因，我们曾经以为能完成这么多，而实际完成的工作与当初预计的还是有区别。"

那个 sprint 里面**差不多**可以完成的故事怎么处理？为什么我们在实际生产率里面没把它的部分故事点数算进来？呵呵，这就突出表现了 Scrum 的要求（实际上也是敏捷软件开发和精益制造的要求）：把事情完全做完！达到可以交付的状态！事情只做了一半，它的价值就是 0（也许还会是负数）。你可以看看 Donald Reinertsen 的"Managing the Design Factory"或是 Poppendieck 的书，从中能够了解更多信息。

那我们在估算生产率的时候，动用了何等神奇的魔力？

火龙果 · 整理
uml.org.cn

有一个很简单的办法：看看团队的历史。看看他们在过去几个 sprint 里面的生产率是多少，然后假定在下一个 sprint 里面生产率差不多不变。

这项技术也被叫做“**昨日天气**（*yesterday's weather*）”。要想使用该技术，必须满足两个条件：团队已经完成了几个 sprint（这样就可以得到统计数据），会以几乎完全相同的方式（团队长度不变，工作状态等条件不变）来进行下一个 sprint。当然也不是绝对如此。

再复杂一点儿，你还可以进行简单的资源计算。假设我们在计划一个 4 人团队 3 星期的 sprint（15 个工作日）。Lisa 会休假两天。Dave 只能投入 50%的时间，另外还会休假一天。把这些加到一起……

……这个 sprint 一共有 50 个可用的人-天。

这是我们的估算生产率么？不！我们估算的单位是**故事点**，差不多可以对应于“理想化的人-天”。一个理想化的人-天是完美、高效、不受打扰的一天，但这种情况太少见了。我们还必须考虑到各种因素，例如要把未计划到的工作添加到 sprint 中、人们患病不能工作等等。

那我们的估算生产率肯定要比 50 少了。少多少呢？我们引入“投入程度（focus factor）”这个词来看一下。

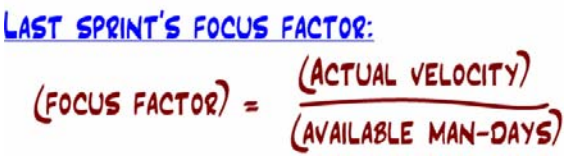
THIS SPRINT'S ESTIMATED VELOCITY:

(AVAILABLE MAN-DAYS) x (FOCUS FACTOR) = (ESTIMATED VELOCITY)

火龙果 · 整理
uml.org.cn

投入程度用来估算团队会在 sprint 中投入多少精力。投入程度低，就表示团队估计会受到很大干扰，或者他们觉得自己的时间估算太过理想化。

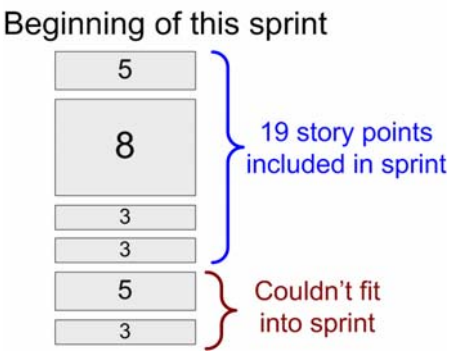
要想得出一个合理的投入程度，最好的办法就是看看上一个 sprint 的值（对前几个 sprint 取平均值自然更好）。



把上一个 sprint 中完成的所有故事的原始估算加起来，得到的和就是**实际生产率**。

假设在上个 sprint 里面，由 Tom，Lisa 和 Sam 组成的 3 人团队在 3 个星期内工作了 45 个人-天，一共完成 18 个故事点。现在我们要为下一个 sprint 估算一下生产率。新伙计 Dave 的加入让情况更复杂了。把假期和新成员算上，我们在下个 sprint 中一共有 50 个人-天。

从上面的公式中可以看出，下个 sprint 的估算生产率是 20 个故事点。这表明团队这个 sprint 中所能做的故事点数之和不能超过 20。



在这种情况下，团队可以选择前 4 个故事，加起来一共 19 个故事点，或者选前 5 个故事，一共 24 个故事点。我们假设他们选了 4 个故事，因为它离 20 最近。如果不太确定，那就再少选些好了。

因为这 4 个故事加起来一共 19 个故事点，所以他们在这个 sprint 中最后的估算生产率就是 19。

“昨日天气”用起来很方便，但需要考虑一些常识。如果上一个 sprint 干得很糟，是因为大部分成员都病了一星期。那你差不多可以放心假设这次运气不会那么坏，给这个 sprint 设个高点的投入程度；如果团队最近刚装了一个执行速度快如闪电的持续集成系统，那你也可以因此提高一下投入程度；如果有新人加入这个 sprint，就得把他的培训占用的精力也算进来，降低投入程度；等等。

只要条件允许，你就应该看看从前的 sprints，计算出平均数，这样可以得到更合理的估算。

如果这是个全新的团队，没有任何数据怎么办？你可以参考一下在类似条件下工作的团队，他们的投入程度数值是多少。

如果没有其他团队可以参考怎么办？随便猜一个数作为投入程度吧。毕竟这个猜测只会在第一个 sprint 里面使用。过了这次以后你就有了历史数据可以分析，然后对投入程度和估算生产率做出不断的改进。

火龙果 · 整理
uml.org.cn

我在新团队中使用的“默认”投入程度通常是 70%，因为这是其他大多数团队都能达到的数值。

我们用的是哪种估算技术？

上面提到了好几种技术——直觉反应、基于昨日天气的生产率计算、基于可用人-天和估算投入程度的生产率计算。

那我们用的是什么？

一般我们都是结合起来用。花不了多大功夫。

我们审视上个 sprint 的投入程度和实际生产率。我们审视这个 sprint 总共可用的资源，估算一个投入程度。我们讨论这两个投入程度之间的区别，必要时进行调整。

大致有了一个要放入 sprint 的故事列表以后，我再进行“直觉反应”的检查。我要求他们暂时忘掉数字，**感觉**一下：在一个 sprint 里一口咬这么多东西会不会难以下咽。如果觉得太多了，那就移走一两个故事。反之亦然。

当天结束以前，只要得出哪些故事要放到 sprint 里面，我们就算完成了目标。投入程度、资源可用性和估算生产率只是用来达成这个目标的手段。

我们为何使用索引卡

在大多数 sprint 计划会议上，大家都会讨论产品 backlog 中的故事细节。对故事进行估算、重定优先级、进一步确认细节、拆分，等等都会在会议上完成。

我们是怎样实际操作的呢？

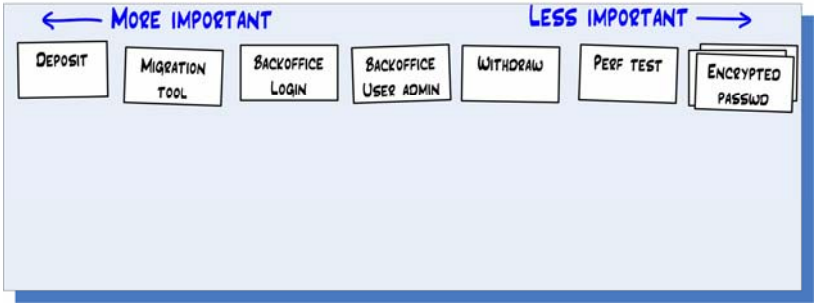
嗯，也许有人认为是这样的。团队打开投影仪，把用 Excel 保存的 backlog 投在墙上，然后一个人（通常是产品负责人或者 Scrum

火龙果 · 整理
uml.org.cn

master）拿过键盘，嘟哝着把一个个故事讲一遍，请大家进行讨论。团队和产品负责人讨论过优先级和具体细节以后，拿着键盘的人会在 Excel 上直接进行修改。

听起来不错？呵，纯粹扯淡。更糟的是，团队一般都是到了会议结束前才**发现**他们一直在扯淡，到最后还没把故事看上一遍呢！

要想收到好的效果，不妨创建**一些索引卡，把它们放到墙上**（或一张大桌子上）。

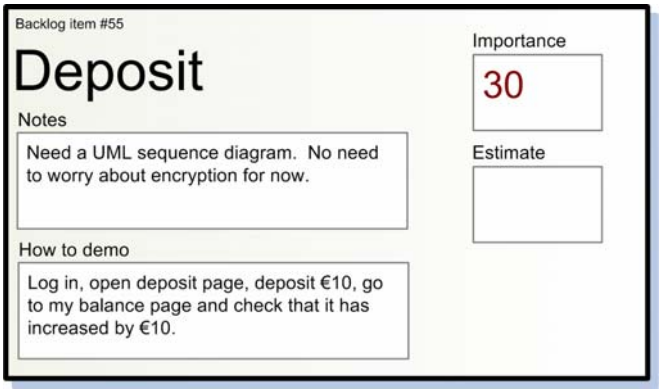


这种用户体验比计算机和投影仪好得多。原因是：

- 大家站起来四处走动=> 他们可以更长时间地保持清醒，并留心会议进展。
- 他们有更多的个人参与感（而不是只有那个拿着键盘的家伙才有）。
- 多个故事可以同时编辑。
- 重新划分优先级变得易如反掌——挪动索引卡就行。
- 会议结束后，索引卡可以拿出会议室，贴在墙上的任务板上（参见第“我们怎样编写 sprint backlogs”）。

你可以手写索引卡（像我们一样），也可以用简单的脚本从产品 backlog 中直接生成可以打印的索引卡。

火龙果 · 整理
uml.org.cn



附——
在我的博客上有这种脚本，地址是http://blog.crisp.se/henrikkniberg.

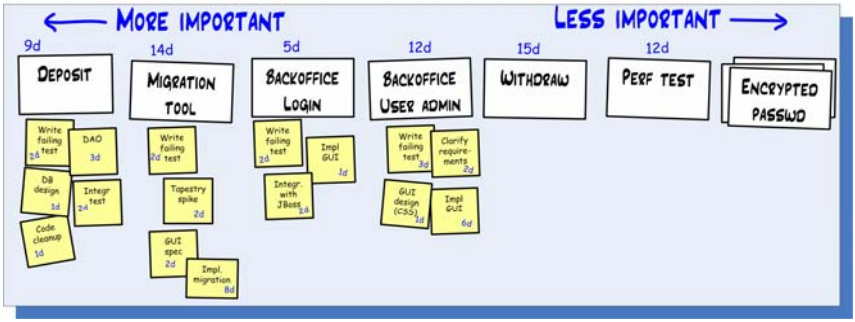
重要事项：Sprint 计划会议结束后，我们的 Scrum master 会手工更新 Excel 中的产品 backlog，以反映故事索引卡中发生的变化。这确实给管理者带来了一点麻烦，但是考虑到用了物理索引卡以后，sprint 计划会议的效率得到了大幅度提高，这种做法还是完全可以接受的。

注意这里的“重要性（Importance）”字段。它和打印时 Excel 中产品 backlog 所记录的“重要性”是一样的。把它放到卡片上，可以帮助我们根据重要性给卡片排序（我们一般把最重要的放到左边，依次向右排列）。不过，一旦卡片被放到墙上，那就可以暂时忽略它的重要性评分，根据它们摆放的相对位置，来对比彼此的重要性。如果产品负责人交换了两张卡片，先不要浪费时间在纸上更新数字，只要确保会议结束后在产品 backlog 做更新就可以。

把故事拆分成任务后，时间估算就变得更容易（也更精确）了。

这个用索引卡来做就又方便又漂亮。你可以把团队分成不同的二人组，让他们每组同时各自拆分一个故事。

我们用即时贴贴在每个故事的下方，每张即时贴表示这个故事中的一个任务。



我们不会让任务拆分出现在产品 backlog 中，原因有二：

- 任务拆分的随机性比较强，在 sprint 进行中，它们常常会发生变化，不断调整，所以保持产品 backlog 的同步很让人头大。
- 产品负责人不需要关心这种程度的细节。

任务拆分的即时贴可以和故事索引卡一起，在 sprint backlog 中被直接重用。（参见“我们怎样编写 sprint backlogs”）。

火龙果 · 整理
uml.org.cn

定义“完成”

有一点很重要：产品负责人和团队需要对“完成”有一致的定义。所有代码被 check in 以后，故事就算完成了吗？还是被部署到测试环境中，经过集成测试组的验证以后才算完成？我们尽可能使用这样的定义：“随时可以上线”，不过有时候我们也这样说：“已经部署到测试服务器上，准备进行验收测试”。

最开始我们用的是比较详细的检查列表。现在我们常说“如果 Scrum 团队中的测试人员说可以，那这个故事就算完成了”。然后责任就到了测试人员身上，他需要保证团队理解了产品负责人的意图，要保证故事的“完成”情况可以符合大家认可的定义。

我们慢慢意识到，不能把所有的故事都一概而论。“查询用户表单”跟“操作指南”这两个故事的处理方式就有很大差异。对后者，“完成”的定义可能就是简单的一句话——“被运营团队认可”。所以说，日常的一些认识往往要好过正式的检查列表。

如果你常常对怎样定义完成感到困惑（就像我们刚开始一样），你或许应该在每个故事上都添加一个字段，起名为“何谓完成”。

使用计划纸牌做时间估算

估算是一项团队活动——通常每个成员都会参与所有故事的估算。为啥要每个人都参加？

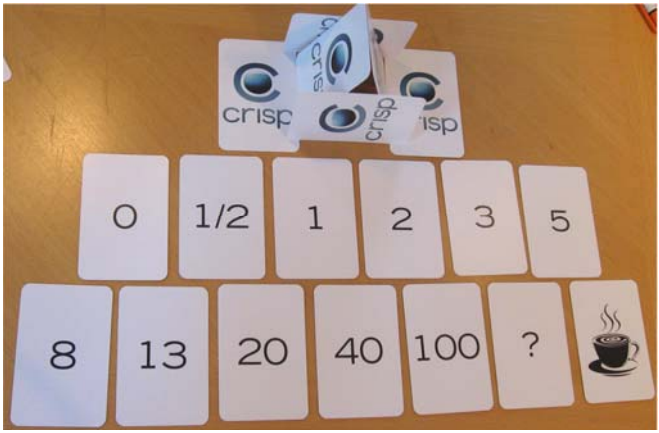
- 在计划的时候，我们一般都还不知道到底谁会来实现哪个故事的哪个部分。
- 每个故事一般有好几个人参与，也包括不同类型的专长（用户界面设计、编程、测试、等等）。
- 团队成员必须要对故事内容有一定的理解才能进行估算。要求每个人都做估算，我们就可以确保他们都理解了每个条目的内容。这样就为大家在 sprint 中相互帮助夯实了基础，也有助于故事中的重要问题被尽早发现。

火龙果 · 整理
uml.org.cn

- 如果要求每个人都对故事做估算，我们就会常常发现两个人对同一个故事的估算结果差异很大。我们应该尽早发现这种问题并就此进行讨论。

如果让整个团队进行估算，通常那个对故事理解最透彻的人会第一个发言。不幸的是，这会严重影响其他人的估算。

有一项很优秀的技术可以避免这一点——它的名字是计划纸牌（我记得是 Mike Cohn 创造出来这个名字的）。



每个人都会得到如上图所示的 13 张卡片。在估算故事的时候，每个人都选出一张卡片来表示他的时间估算（以故事点的方式表示），并把它正面朝下扣在桌上。所有人都完成以后，桌上的纸牌会被同时揭开。这样每个人都会被迫进行自我思考，而不是依赖于其他人估算的结果。

如果在两个估算之间有着巨大差异，团队就会就此进行讨论，并试图让大家对故事内容达成共识。他们也许会进行任务分解，之后再重新估算。这样的循环会往复进行，直到时间估算趋于一致为止，也就是每个人对这个故事的估算都差不多相同。

火龙果 · 整理
uml.org.cn

重要的是，我们必须提醒团队成员，他们要对这个故事中所包含的全部工作进行估算。而不是“他们自己负责”的部分工作。测试人员不能只估算测试工作。

注意，这里的数字顺序不是线性的。例如在 40 和 100 之间就没有数字。为什么这样？

这是因为，一旦时间的估算值比较大，其精确度就很难把握；这样做就可以避免人们对估算精确度产生错误的印象。如果一个故事的估算值是差不多 20 个故事点，它到底应该是 20 还是 18 还是 21，其实无关紧要。我们知道的就是它是一个很大的故事，很难估算。所以 20 只是一个粗略估计。

需要进行更精确的估算？那就把故事分拆，去估算那些更小的故事！

另外，你也不能搞那种把 5 和 2 加起来得到 7 的把戏。要么选 5，要么选 8，没有 7。

有些卡片比较特殊：

- 0 = “这个故事已经完成了”或者“这个故事根本没啥东西，几分钟就能搞定”。
- ? = “我一点概念都没有。没想法。”
- 咖啡杯 = “我太累了，先歇会吧。”

明确故事内容

在 sprint 演示会议上，团队自豪地演示了一个新特性，但产品负责人却皱起眉头，“呃，看上去不错，但**这不是我要的！**”发生这种事情可真是糟透了！

怎样才能让产品负责人和团队对故事有同样的理解？或者保证所有的团队成员对每个故事都有同样的理解？嗯，这可没法做到。不过还是有些简单技术，可以识别出最明显的误解。最简单的办法就

火龙果 · 整理
uml.org.cn

是确保每个故事的所有字段都被填满（更精确地说，这里提到的是具有高优先级，应该在这个 sprint 里面完成的故事）。

例 1：
团队和产品负责人都对 sprint 计划很满意，打算结束会议。这时 Scrum master 问了一个问题，“等一下，还有个‘添加用户’的故事没有估算时间呢，把它解决了吧！”几轮计划纸牌以后，团队意见达成一致，认为这个故事需要 20 个故事点；产品负责人却站了起来，说话因为生气也走了调：“什、什、什么？！”经过几分钟的激烈争吵，最后发现是团队错误理解了“增加用户”这个故事的范围，他们以为这表示“要有个漂亮的 web 界面来添加、删除、移除和查询用户”，但是产品负责人只是想“通过手写 SQL 操作数据库来添加用户”。他们重新进行评估，给它 5 个故事点，达成共识。

例 2：
团队和产品负责人都对 sprint 计划很满意，打算结束会议。这时 Scrum master 问了一个问题，“等一下，还有一个‘添加用户’的故事，它怎么演示呢？”一阵窃窃私语之后，某人站起来说，“呃，首先我们登录 Web 站点，然后……”产品负责人打断了他的话，“登录 Web 站点？！不不不，这个功能跟 Web 站点一点关系都没有，你给技术管理员提供个傻瓜都能用的 SQL 脚本就行。”

“如何演示”这段描述可以（而且应该）**非常精简**！不然你就没法按时结束 sprint 计划会议。基本上，它就是用最平白浅显的语言，来描述如何手工执行最典型的测试场景。“先这样，然后那样，最后验证这一点。”

我发现，使用这种简单的描述，**常常**能够发现对于故事范围的最严重的误解。这种事发现的越早越好，不是么？

把故事拆分成更小的故事

故事不应该太短，也不应太长（从估算的角度出发）。如果你有一大堆 0.5 个故事点的故事，那你恐怕就会成为微观管理的受害者了。

火龙果 · 整理
uml.org.cn

与之相反，40 个点的故事，到最后很可能只能**部分完成**，这样不会为公司带来任何价值，只会增加管理成本。进一步来说，如果你们估算的生产率是 70，而最高优先级的两个故事都是 40 个故事点，那做计划可就有麻烦了。

摆在团队面前的只有两种选择：要么只选一个条目，完成当初允诺的工作后，还有不少空闲时间，导致承诺不足（under-committing）；要么两个条目都选上，最后无法完成当初允诺的工作量，导致过度承诺（over-committing）。

我发现很大的故事基本上都能进行拆分。只要确定每个小故事依然可以交付业务价值就行。

我们常常都力求保证故事的大小在 2 至 8 个人-天之间。一个普通团队的生产率大约是 40-60，所以大概每个 sprint 可以完成 10 个故事。有时会减少到 5 个，有时也会多到 15 个。处在这个数量范围之间的索引卡是比较容易管理的。

把故事拆分成任务

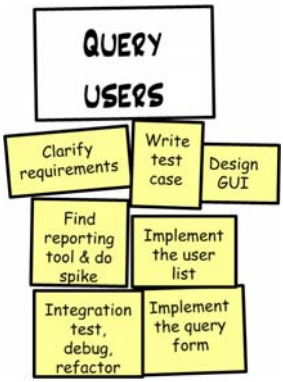
等一下。“任务”和“故事”的区别是什么呢？嗯，这个问题问得不错。

区别很简单。故事是可以交付的东西，是产品负责人所关心的。任务是不可交付的东西，产品负责人对它也不关心。

在下图的例子中，故事被拆分成更小的故事：

下面是把故事拆分成任务的例子：

火龙果 · 整理
uml.org.cn



我们会看到一些很有趣的现象：

- 新组建的 Scrum 团队不愿意花时间来预先把故事拆分成任务。有些人觉得这像是瀑布式的做法。
- 有些故事，大家都理解得很清楚，那么预先拆分还是随后拆分都一样简单。
- 这种类型的拆分常常可以发现一些会导致时间估算增加的工作，最后得出的 sprint 计划会更贴近现实。
- 这种预先拆分可以给每日例会的效率带来显著提高（参见“我们怎样进行每日例会”）。
- 即使拆分不够精确，而且一旦开始具体工作，事先的拆分结果也许会发生变化，但我们依然可以得到以上种种好处。

所以，我们试着把 sprint 计划会议的时间放到足够长，保证有时间进行任务拆分，但如果时间不够的话，我们就不做了。（参加下面的“最后界限在哪里”）。

注意——我们在实践 TDD（测试驱动开发），所以几乎每个故事的第一个任务都是“编写一个失败的测试”，而最后一个任务是“重构”（提高代码的可读性，消除重复）。

火龙果 · 整理
uml.org.cn

定下每日例会的时间地点

Sprint 计划会议有一个产物常常被人们忽略："确定的时间和地点，以供举办每日例会"。没有这一点，你的 sprint 就会有个"开门黑"。实际上，每个人都是在当前 sprint 的第一个每日例会上决定怎样开始工作。

我喜欢早上开会。不过我得承认，我们没有真正试过在下午或者中午进行每日例会。

在下午开每日例会的缺点：早上来工作的时候，你必须试着记起来你昨天对别人说过今天要做什么。

在上午开每日例会的缺点：早上来工作的时候，你必须试着记起来你昨天做了些什么，这样才能跟别人讲。

我的看法是，第一个缺点更糟，因为最重要的事情是你**打算干**什么，而不是**已经干了**什么。

我们的默认做法是选一个大家都不会有异议的最早时间。一般是 9:00，9:30 或者 10:00。最关键的是，这必须是每个人都能完全接受的时间。

最后界限在哪里

OK，现在时间已经用完了。如果时间不够的话，那么我们该把哪些本该做的事情砍掉呢？

嗯，我总是用下面这个优先级列表：

优先级 1：sprint 目标和演示日期。这是启动 sprint 最起码应该有的东西。团队有一个目标，一个结束日期，然后就可以马上根据产品 backlog 开始工作。没错，这是不像话，你应该认真考虑一下明天再开个新的 sprint 计划会议。不过如果确实需要马上启动 sprint，不妨先这么着吧。认真说来，只有这么点儿信息就开始 sprint，我还从来没有试过。

火龙果 · 整理
uml.org.cn

优先级 2：经过团队认可、要添加到这个 sprint 中的故事列表。

优先级 3：Sprint 中每个故事的估算值。

优先级 4：Sprint 中每个故事的“如何演示”。

优先级 5：生产率和资源计算，用作 sprint 计划的现实核查。包括团队成员的名单及每个人的承诺（不然就没法计算生产率）。

优先级 6：明确每日例会固定举行的时间地点。这只需要花几分钟，但如果时间不够用，Scrum master 可以在会后直接定下来，邮件通知所有人。

优先级 7：把故事拆分成任务。这个拆分也可以在每日例会上做，不过这会稍稍打乱 sprint 的流程。

技术故事

这有个很复杂的问题：技术故事。或者叫做非功能性条目，或者你想叫它什么都行。

我指的是需要完成但又不属于可交付物的东西，跟任何故事都没有直接关联，不会给产品负责人带来直接的价值。

我们把它叫做“技术故事”。

例如：

- **安装持续构建服务器**
  - 为什么要完成它：因为它会节省开发人员的大量时间，到迭代结束的时候，集成也不太容易出现重大问题。
- **编写系统设计概览**
  - 为什么要完成它：因为开发人员常常会忘记系统的整体设计，写出与之不一致的代码。团队需要有个

火龙果 · 整理
uml.org.cn

描述整体概况的文档，保证每个人对设计都有同样的理解。

- **重构 DAO 层**
  - 为什么要完成它：因为 DAO 层代码已经乱成一团了。混乱带来了本可以避免的 bug，每个人的时间都在被无谓的消耗。清理代码可以节省大家的时间，提高系统的健壮性。
- **升级 Jira** （bug 跟踪工具）
  - 为什么要完成它：当前的版本 bug 狂多，又很慢，升级以后可以节省大家时间。

按照一般的观点来看，这些算是故事吗？或者是一些跟任何故事都没有直接关联的任务？谁来给它们划分优先级？产品负责人应该参与其中吗？

我们尝试过各种处理方式。我们曾经把它们跟别的故事一样，当作第一等的故事。但这样并不好。产品负责人来对产品 backlog 划分优先级的时候，就像在拿苹果跟桔子作对比一样。实际上，出于显而易见的原因，技术故事常常会因为某种原因给设置一个低优先级，例如："嘿，兄弟们，我知道持续构建服务器很重要，不过让我们先来完成一些可以带来收入的特性吧，然后再来弄那些技术上的东西，行不？"

有些时候，产品负责人的做法是对的，但这只是少数情况。我们得出的结论是，产品负责人往往不能对此做出正确的权衡。所以我们采取了下面这些做法：

1) 试着避免技术故事。努力找到一种方式，把技术故事变成可以衡量业务价值的普通故事。这样有助于产品负责人做出正确的权衡。
2) 如果无法把技术故事转变成普通故事，那就看看这项工作能不能当作另一个故事中的某个任务。例如，"重构 DAO 层"可以作为"编辑用户"中的一个任务，因为这个故事会涉及到 DAO 层。

火龙果 · 整理
uml.org.cn

3) 如果以上二者都不管用，那就把它定义为一个技术故事，用另外一个单独的列表来存放。产品负责人能看到它，但是不能编辑它。用“投入程度”和“预估生产率”这两个参数来跟产品负责人协商，从 sprint 里拨出一些时间来完成这些技术故事。

下面是一个示例（这段对话跟我们某个 sprint 计划会议中发生过的一幕似曾相识）。

- **团队：**“我们要完成一些内部的技术工作。也许要从我们的时间里抽出来 10%，也就是把投入程度从 75%降低到 65%，你看行吗？”
- **产品负责人：**“不行！我们没那个时间了！”
- **团队：**“嗯……那看看上一个 sprint 吧（大家的头全都转向了白板上的生产率草图）。我们估算的生产率是 80，但实际才有 40，没错吧？”
- **产品负责人：**“没错！所以我们没时间干那些内部的技术活了！我们需要新功能！”
- **团队：**“呃，我们的生产率变得这么糟糕的**原因**就是，构造可以测试的稳定版本占用了太多时间。
- **产品负责人：**“嗯，那然后呢？”
- **团队：**“唔，如果我们不做点什么的话，生产率还会继续这么烂下去。”
- **产品负责人：**“嗯，接着说？”
- **团队：**“所以我们建议在这个 sprint 里抽出大概 10%的时间来搭一个持续构建服务器，完成相关的一些事情，这样就不会再受集成的折磨。接下来，每个 sprint 里面，我们的生产率都会提高**至少** 20%！”
- **产品负责人：**“啊？真的吗？那为什么上个 sprint 我们没这么干？！”
- **团队：**“嗯……因为你不同意……”
- **产品负责人：**“哦，嗯……那好吧，这主意听上去不错，开始干吧！”

火龙果 · 整理
uml.org.cn

当然，还可以把产品负责人排除在外，或者是告诉他一个不可协商的投入程度。但你不妨先**尝试**一下，让你们的想法达成一致。

如果产品负责人能力比较强，也能听进别人的意见（这一点上，我们比较幸运），那我建议你最好还是尽量让他知道所有的事情，让他制定一切工作的优先级。透明也是 Scrum 的核心价值，不是吗？

Bug 跟踪系统 vs. 产品 backlog

这个问题有点难搞。用 Excel 来管理产品 backlog 很不错，不过你仍然需要一个 bug 跟踪系统，这时 Excel 就无奈了。我们用的是 Jira。

那我们怎么把 Jira 上的 issue 带到 sprint 计划会议上来呢？我的意思是，如果无视这些 issue，只关心故事，这可没什么好处。

我们试过几种办法：

1) 产品负责人打印出 Jira 中优先级最高的一些条目，带到 sprint 计划会议中，跟其他故事一起贴到墙上（因此就暗暗地指明了这些 issue 相对其他故事的优先级）。
2) 产品负责人创建一些指向 Jira 条目的故事。例如“修复那几个后台报表最严重的 bug，序号是 Jira-124、Jira-126，还有 Jira-180”。
3) 修复 bug 被当作 sprint 以外的工作，也就是说，团队会保持一个足够低的投入程度（例如 50%），从而保证他们有时间来修复 bug。然后我们就可以简单假设，在每一个 sprint 中，团队都会用一些时间来修复 Jira 上报告的 bug。
4) 把产品 backlog 放到 Jira 上（也就是放弃 Excel）。把 bug 与其他故事同等看待。

我们没有发现哪种策略最适合我们，实际来看，团队与团队、sprint 与 sprint 之间的做法都会有差异。不过我还是倾向于使用第一种方案。既简单，效果又好。

火龙果 · 整理
uml.org.cn

Sprint 计划会议终于结束了！

哇塞，没想到这一章会这么长！我想这应该可以表示出我的个人观点——sprint 计划会议是 Scrum 中最重要的活动。在这里投入大量努力，保证它顺利完成，后面的工作就会轻松很多。

如果每个人（所有的团队成员和产品负责人）离开会场时都面带微笑，第二天醒来时面带微笑，在第一次的每日例会上面带微笑，那 sprint 计划会议就是成功的。

当然，什么事情都有可能出现问题，但至少你不能归咎于 sprint 计划:o)

火龙果 · 整理
uml.org.cn

5

我们怎样让别人了解我们的 sprint

我们要让整个公司了解我们在做些什么，这件事情至关重要。否则其他人就会发出抱怨，甚或对我们的工作做出臆断。

我们为此使用“sprint 信息页”。

Jackass team, sprint 15

Sprint goal
- Beta-ready release!

Sprint backlog (estimates in parenthesis)
- Deposit (3)
- Migration tool (8)
- Backoffice login (5)
- Backoffice user admin (5)

Estimated velocity: 21

Schedule
- Sprint period: 2006-11-06 to 2006-11-24
- Daily scrum: 9:30 – 9:45, in the team room
- Sprint demo: 2006-11-24, 13:00, in the cafeteria

Team
- Jim
- Erica (scrum master)
- Tom (75%)
- Eva
- John

火龙果 · 整理
uml.org.cn

有时我们会把每个故事该如何演示也包括进来。

Sprint 计划会议一结束，Scrum master 就创建这个页面，把它放到 wiki 上，给整个公司发一个“垃圾”邮件。

```
Subject: Jackass sprint 15 started

Hi all! The Jackass team has now started sprint
15. Our goal is to demonstrate a beta-ready
release on nov 24.

See the sprint info page for details:
http://wiki.mycompany.com/jackass/sprint15
```

我们在 wiki 上还有个“dashboard”页面，链向所有正在进行的 sprint。

Corporate Dashboard

Ongoing sprints

- Team X sprint 15
- Team Y sprint 12
- Team Z sprint 1

此外，Scrum master 还会把 sprint 信息页打印出来，贴到团队房间外面的墙上。路过的每个人都可以阅读这张纸，了解这个团队所做的事情。因为其中还包括了每日例会的时间地点，所以他也能知道到哪里去了解更多信息。

Sprint 接近尾声时，Scrum master 会把即将来临的演示告知每个人。

火龙果 · 整理
uml.org.cn


```
Subject: Jackass sprint demo tomorrow at 13:00 in the cafeteria.

Hi all! You are welcome to attend our sprint demo at 13:00 in
the cafeteria tomorrow (friday). We will demonstrate a
beta-ready release.

See the sprint info page for details:
http://wiki.mycompany.com/jackass/sprint15
```

有了这一切以后，就没人还能找借口说**不知道**你们的工作状态了。

火龙果 · 整理
uml.org.cn

6

我们怎样编写 sprint backlog

你已经走了这么远了？嗯，干得好。

现在我们已经完成了 sprint 计划会议，整个世界都了解了我们下一个熠熠生辉的 sprint。Scrum master 现在应该创建 sprint backlog 了。它应该在 sprint 计划会议**之后**，第一次每日例会**之前**完成。

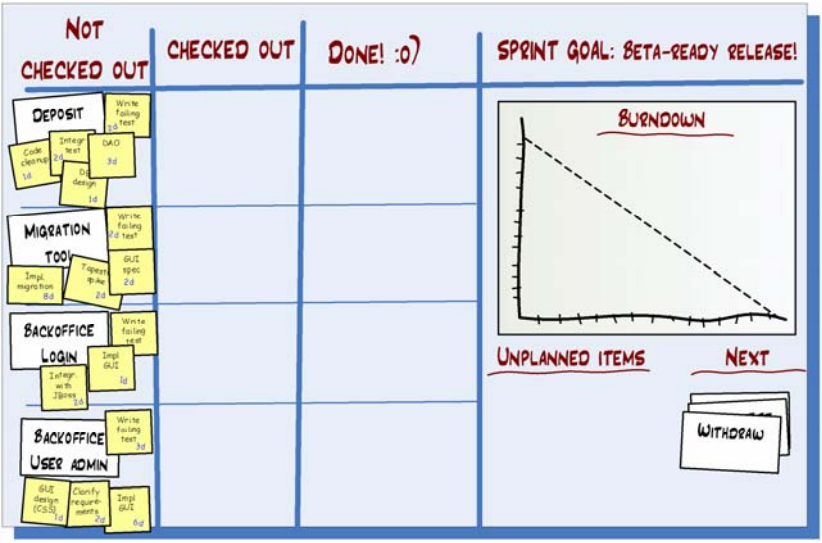
Sprint backlog 的形式

我们曾经尝试过用多种形式来保存 sprint backlog，包括 Jira、Excel，还有挂在墙上的任务板。开始我们主要使用 Excel，有很多公开的 Excel 模板可以用来管理 sprint backlog——包括自动生成的燃尽图等等。在如何改良基于 Excel 的 sprint backlog 方面，我有很多想法，但此处暂且不提，我也不会在这里举例。

下面要仔细描述的，是我们发现管理 sprint backlog 最有效的形式——挂在墙上的任务板！

找一面尚未使用或者充满无用信息（如公司 logo、陈旧图表或者丑陋的涂鸦）的大墙。清理墙壁（除非不得已才去请求别人许可）。在墙上贴上一张很大很大的纸（至少 2x2 平方米，大团队需要 3x2 平方米）。然后这样规划：

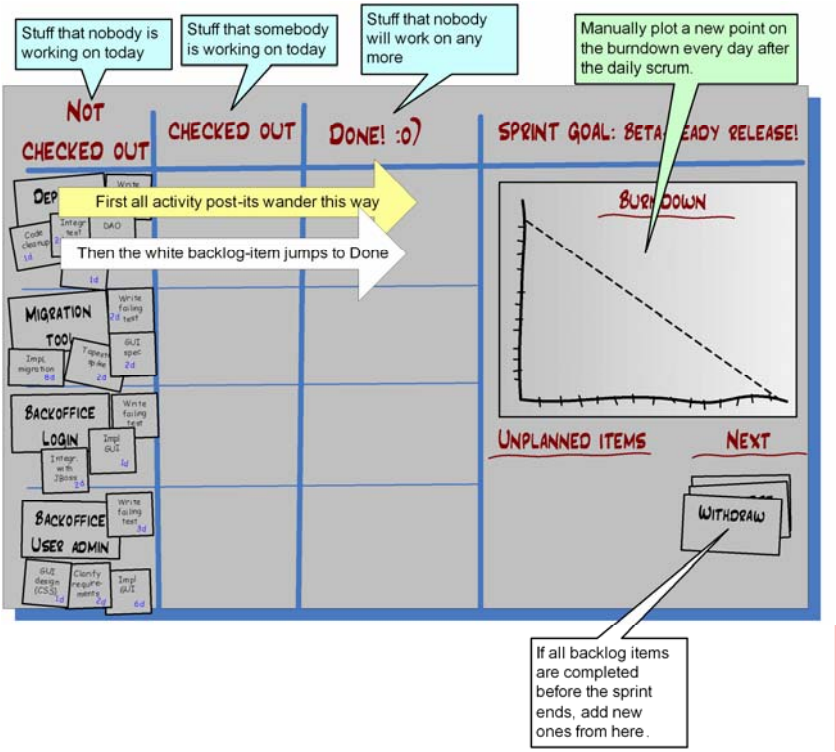
火龙果 · 整理
uml.org.cn



你当然也可以用白板。不过那多少有点浪费。可能的话，还是把白板省下来画设计草图，用没有挂白板的墙做任务板。

注意——如果你用贴纸来记录任务，别忘了用真正的胶带把它们粘好，否则有一天你会发现所有的贴纸都在地上堆成一堆。

任务板怎样发挥作用



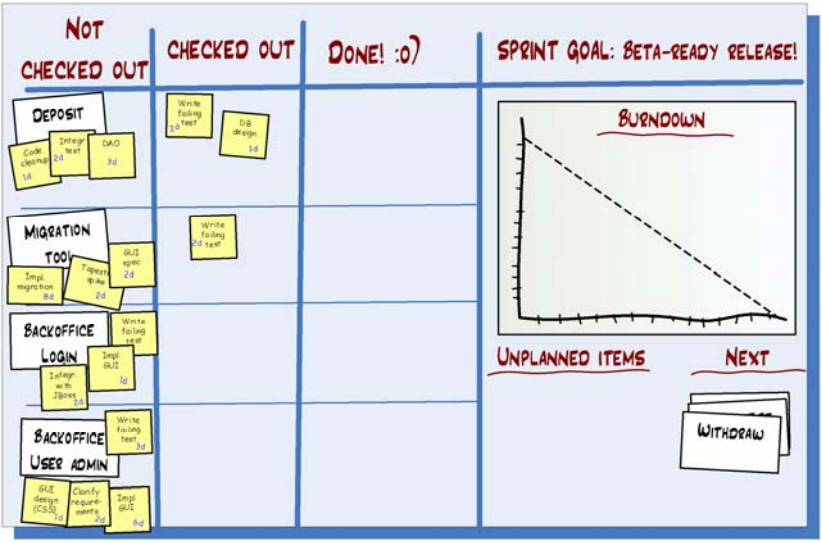
当然，你也可以另外添上许多列，比如“等待集成测试”，或者“已取消”。但是在把这一切搞复杂之前，请试着仔细考虑考虑，你要添上去的那一列真的，**真的**是没它不行吗？

我发现在处理这种类型的事情时，“简单性”会发挥极大的作用；所以除非不这样做会付出极大代价，我才愿意让事情变得更加复杂。

例 1 ——首次每日 scrum 之后

在首次每日例会以后，任务板可能会变成这样：

火龙果 · 整理
uml.org.cn

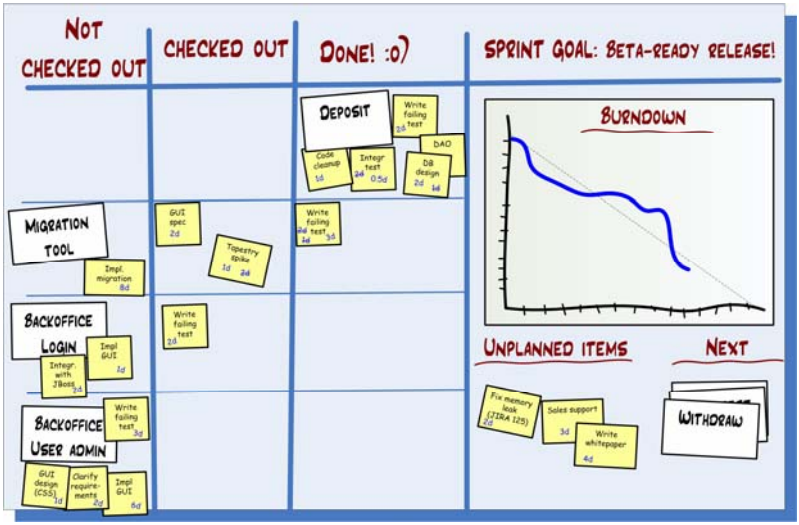


你可以看到，有三个任务已经被“checked out”，团队今天将处理这些条目的工作。

在大团队中，有时某个任务会一直停留在“checked out”状态，因为已经没人记得是谁认领了这个任务。要是这种情况一再发生，他们就会在任务上加上标签，记录谁 check out 了这个任务。

例 2 ——几天以后

几天以后，任务板可能会变成这样：



你可以看到，我们已经完成了“DEPOSIT”这个故事（它已经被签入了源代码仓库，经过了测试、重构等等步骤）。“MIGRATION TOOL”只完成了一部分，“BACKOFFICE LOGIN”刚刚开始，“BACKOFFICE USER ADMIN”还没有开始。

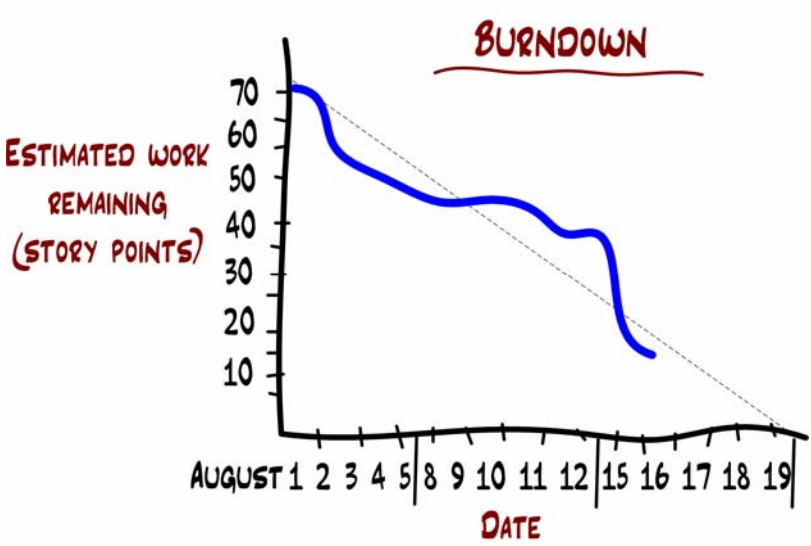
我们有 3 个未经过计划的条目，放在任务板的右下角。进行 sprint 回顾的时候应当记住这一点。

下图是一个真实的 sprint backlog。这里 sprint 已经接近结尾。在 sprint 的进展中，这张表变得相当乱，不过因为这个状态很短，所以没太大关系。每个新的 sprint 启动后，我们都会创建一个全新的、干净的 sprint backlog。

火龙果 · 整理
uml.org.cn

燃尽图如何发挥作用

让我们把目光投向燃尽图：



这张图包含的信息有：

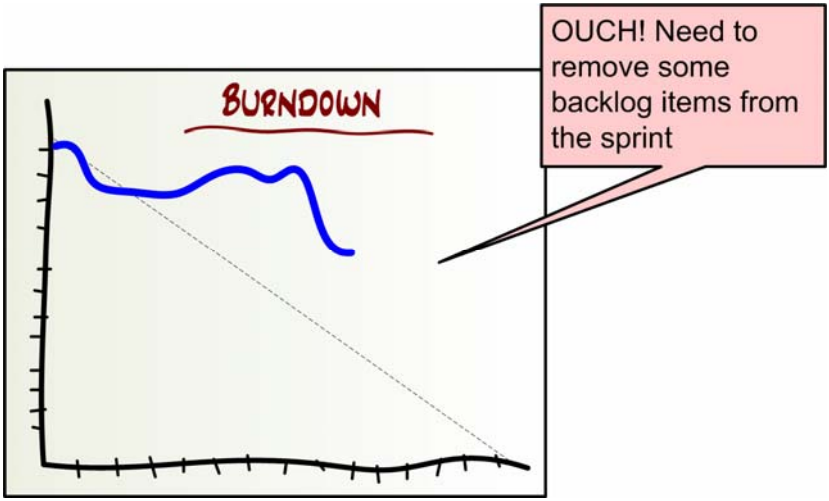
- Sprint 的第一天，8 月 1 号，团队估算出剩下 70 个故事点要完成。这实际上就是整个 sprint 的**估算生产率**。
- 在 8 月 16 号，团队估算出还剩下 15 个故事点的任务要做。跟表示趋势的虚线相对比，团队的工作状态还是差不多沿着正轨的。按照这个速度，他们能在 sprint 结束时完成所有任务。

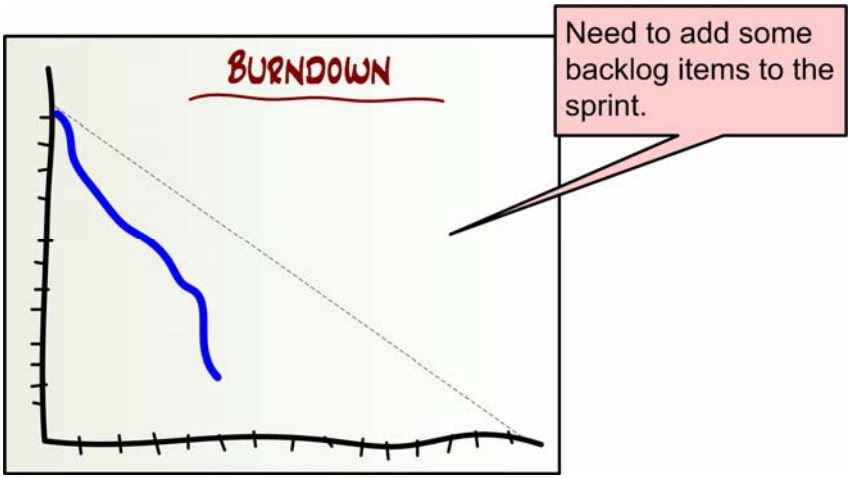
我们没把周末放到表示时间的 x 轴上，因为很少有人会在周末干活儿。我们曾经把周末也算了进来，但是这两天的曲线是平的，看上去就像警告 sprint 中出现了问题，这就让人看着不爽了。

火龙果 · 整理
uml.org.cn

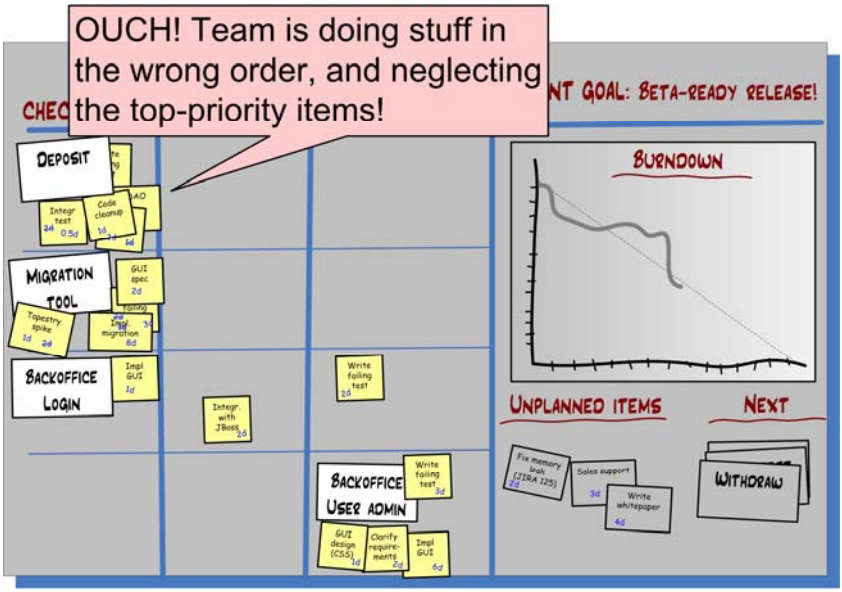
任务板警示标记

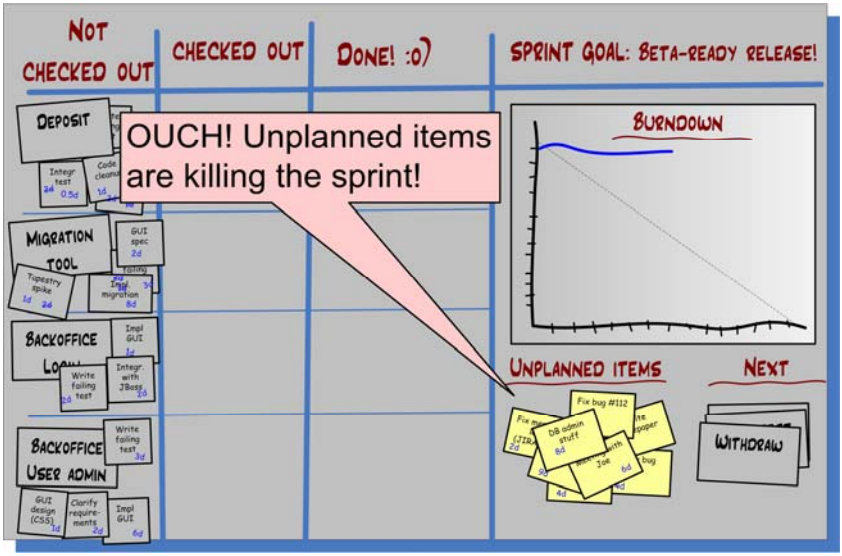
在任务板上匆匆一瞥，就可以大致了解到 sprint 的进展状态。Scrum master 应当确保团队会对这些警示标记做出反应：





火龙果 · 整理
uml.org.cn




NOT CHECKED OUT
CHECKED OUT
DONE! :o)
SPRINT GOAL: BETA-READY RELEASE!
OUCH! Unplanned items are killing the sprint!
DEPOSIT
MIGRATION TOOL
BACKOFFICE
BACKOFFICE USER ADMIN
BURNDOWN
UNPLANNED ITEMS
NEXT
WITHDRAW

火龙果 · 整理
uml.org.cn

嘿，该怎样进行跟踪呢？

在这种模型中，如果必须跟踪的话，那我能提供的最佳方式，就是每天给任务板拍一张数码照片。我有时也这样干，但一直没用到这些照片。

如果你确实需要跟踪任务进度，任务板这种解决方案可能就不太适合你。

不过我建议你应该试着去评估一下，对 sprint 进行细节跟踪能带给你多大价值。Sprint 完成以后，可以工作的代码已被交付，文档也被 check in，那还有谁会真的关心 sprint 的第 5 天完成了多少故事呢？又有谁会真的关心“为 Deposit 编写失败测试”曾经的估算量是多少？

天数估算 vs. 小时估算

在讲述 Scrum 的书和文章中，大多数都是用小时而不是天数来估算时间。我们也这样干过。我们的通用方程为 1 个有效的人-天=6 个有效的人-小时。

现在我们已经不这么干了，至少在大部分团队中如此。原因在于：

- 人-小时的粒度太细了，它会导致太多小到 1-2 个小时的任务出现，然后就会引发微观管理。
- 最后发现实际上每个人还是按照人-天的方式来思考，只是在填写数据时把它乘 6 就得到了人-小时。“嗯……这个任务要花一天。哦对，我要写小时数，那我就写 6 小时好了。”
- 两种不同的单位会导致混乱。“这个估算的单位是啥？人-天还是人-小时？”

所以现在我们用人-天作为所有时间估算的基础（虽然我们也把它叫做故事点）。它的最小值是 0.5，也就是说小于 0.5 的任务要么被移除，要么跟其他任务合并，要么就干脆给它 0.5 的估算值（稍稍超出估算不会带来很大影响）。干净利落。

火龙果 · 整理
uml.org.cn

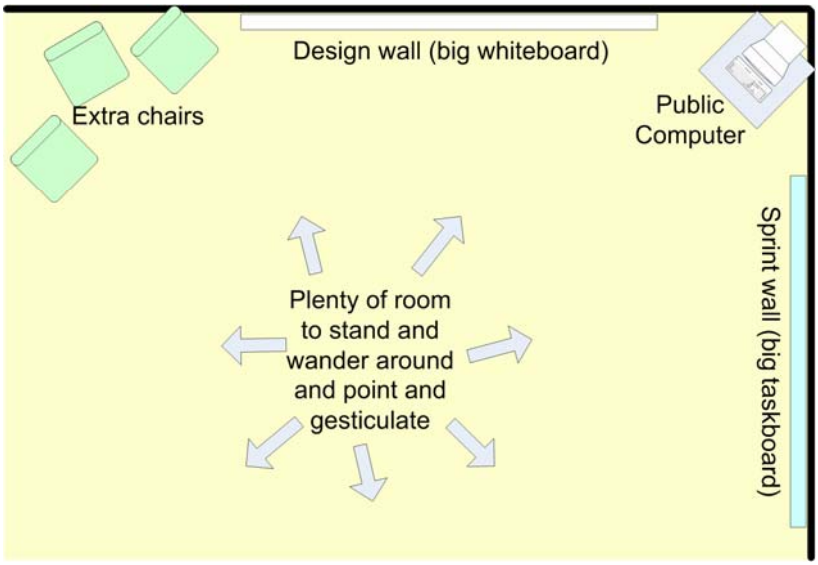
7

我们怎样布置团队房间

设计角

我曾发现这样一个事实：大多数最有趣最有价值的设计讨论，都是在任务板前面自然而然地发生的。

所以我们试着把这个区域布置成一个明显的“设计角”。



火龙果 · 整理
uml.org.cn

这着实很有用。要得到系统概况，不妨站到设计角前面看看墙上的文字和图表，然后回到计算机前面用最近一次的系统构建结果尝试一下，还有什么方式能比这更有效呢？（如果你运气不错、拥有持续构建的话。参见“我们怎样合并 Scrum 和 XP”）。

“设计墙”只是一块大白板，上面画着最重要的设计草图，还有打印出来的、最重要的设计文档（顺序图，GUI 原型，领域模型等等）

上图为在上述角落中进行的每日例会。

嗯……这个燃尽图看起来太干净了，线条也很直，不过这个团队坚持说它是真实情况的反映:o)

火龙果 · 整理
uml.org.cn

让团队坐在一起！

在安排座位、布置桌椅这方面，有一件事情怎么强调也不为过。

让团队坐在一起！

说的更清楚一点，我说的是

让团队坐在一起！

大家都懒的动。至少我工作的地方是这样的。他们不想收拾起自己的零碎儿、拔下计算机电源、把东西都挪到新的电脑桌上，然后把一切再插回去。挪的距离越短，这种抵触情绪就越强烈。“老大，干嘛呢，动这 5 米有啥用？”

但是为了把 Scrum 团队弄得上档次一些，在这方面没有其它选择。一定要让他们坐到一起。即使你不得不私下里威胁每一个人，给他们清理零碎，把老位子收拾利索。如果空间不够，那就找个地方创造空间。就算把团队搬到地下室里面去也在所不惜。把桌子拼到一起，贿赂办公室管理员，竭尽所能。只要能让他们坐到一起。

只要让他们坐到一起，就会有立竿见影的成效。过上一个 sprint，团队就会认为挪到一起是绝妙的主意（从我的个人经验来看，你的团队也有可能会固执地不承认这一点）。

那怎么才算坐到“一起”？桌子该怎么摆？呃，我在这方面没太多建议。而且就算我有，恐怕大多数团队也没有奢侈到可以决定怎么摆放桌子。工作空间中总是有太多物理限制——隔壁的团队、厕所的门、屋子中间的大型自动售货机，等等。

“一起”意味着：

- **互相听到：**所有人都可以彼此交谈，不必大声喊，不必离开座位。

火龙果 · 整理
uml.org.cn

- **互相看到：**所有人都可以看到彼此，都能看到任务板——不用非得近到可以**看清楚**内容，但至少可以**看到个大概**。
- **隔离：**如果你们整个团队突然站起来，自发形成一个激烈的设计讨论，团队外的任何人都不会被打扰到。反之亦然。

“隔离”并不是意味着这个团队需要被完全隔离起来。在一个格子间的环境中，如果你的团队拥有自己的格子，而且隔间的墙足够大，可以屏蔽墙内外的**大多数**噪音，这也就足够了。

如果是分布式团队怎么办？呃，那就没辙了。多使用一些技术辅助手段来减少分布式带来的损害吧——比如视频会议、网络摄像头、桌面共享工具，等等。

让产品负责人无路可走

产品负责人应该离团队很近，既方便团队成员走过来讨论问题，他也能随时踱到任务板前面去。但是他不应该跟团队坐在一起。为什么？因为这样他就无法控制自己不去关注具体细节，团队也无法“凝结”成整体（即达到关系紧密、自组织、具有超高生产力的状态）。

说实话，这全是我自己的猜测。因为从来没有碰到过产品负责人跟团队坐到一起的情况，所以上面的说法也没有实际根据。只是一种直观的感觉，也有从其他 Scrum Master 那里的道听途说。

火龙果 · 整理
uml.org.cn

让经理和教练无路可走

这一刻，手指的移动变得格外艰难，因为我既是经理，又是教练……

尽可能和团队紧密工作，这是我的职责。我组建团队、在团队间切换、跟人结对编程、培训 Scrum master、组织 sprint 计划会议……。事后想想，大多数人都认为这是个**好事情**，因为我在敏捷软件开发方面具有相当的经验。

但是，另一方面，我同时（“星球大战”中黑武士的出场音乐响起）也是开发主管，从行政上担任着经理职务。于是每次我来到团队中间，他们的自组织性就会降低。“见鬼，老大来了，他可能有很多想法，告诉我们应该干啥，谁应该去做什么。让他说吧。”

我是这么看的，如果你是 Scrum 教练（或许同时也是经理），就应该尽可能贴近团队。但不久以后，就离开他们，让他们凝聚在一起，自我管理。然后每隔一段时间（不要太频繁），就去参加一次他们的 sprint 演示，看看任务板，听听晨会。如果发现有可以改进的地方，就把 Scrum master 拽出来指导他。但是**不要**在团队面前这样干。另外，如果团队足够信任你，他们不会看见你就闭上嘴巴，那去参加他们的 sprint 回顾也是个好主意（参见“我们怎样做 sprint 回顾”）。

对于运转良好的 Scrum 团队，只需要保证他们可以得到一切所需的东西，然后就可以任他们自由发挥了（除了 sprint 演示以外）。

火龙果 · 整理
uml.org.cn

8

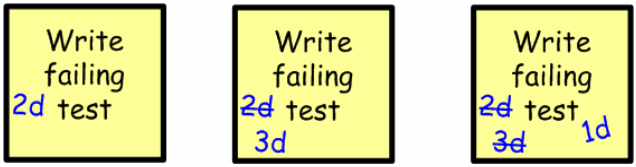
我们怎样进行每日例会

我们的每日例会跟书中的几乎没啥两样。它们每天都会在同一个地方，同一个时间进行。最开始，我们都是去一个单独的房间做 sprint 计划（当我们还是用电子版 sprint backlog 的时候），不过现在我们都是在团队房间里、任务板的前面进行每日例会。没什么能比它效果更好。

一般我们都是开站立会议，以防止持续时间超过 15 分钟。

我们怎样更新任务板

一般我们都是在每日例会的时候更新任务板。每个人都会一边描述昨天已经做的事情和今天要做的事情，一边移动任务板上对应的即时贴。如果他讲的是一个未经计划的条目，那他就新写一张即时贴，贴到板上。如果他更新了时间估算，那就在即时贴上写上新的时间，把旧的划掉。有时候 Scrum master 会在大家讲述各自工作的同时做这件事情。



有些团队会规定每个人都要在会议**开始前**更新任务板。这个做法也不错。你只要订好规则，坚持执行就行。

火龙果 · 整理
uml.org.cn

但无论你的 sprint backlog 是什么形式，都要尽力让**整个团队**参与到保持 sprint backlog 及时更新的工作中来。我们曾经试过让 Scrum master 自己维护 sprint backlog，他就不得不每天都去询问大家各自剩余的工作估算时间。这种做法的缺点是：

- Scrum master 把太多时间用在了管理之类的工作上，而不是为团队提供支持，消除他们的障碍。
- 因为团队成员不再关心 sprint backlog，所以他们就意识不到 sprint 的状态。缺少了反馈，团队整体的敏捷度和精力的集中程度都会下降。

如果 sprint backlog 设计得很好，那每个人都应该很容易修改它。

每日例会一结束，就要有人算出剩余工作的时间估算之和（自然，那些被放在“完成”一栏中的就不用算了），在 sprint 燃尽图上画上一个新的点。

处理迟到的家伙

有些团队会弄个存钱罐。如果你来晚了，即使只迟到一分钟，也必须往里面投入定额的钱。没有人关心迟到的理由。即使你提前打电话声明会迟到，那也得被罚款。

除非你有很好的理由，比如预约了看病，或者是举办你自己的婚礼等等。

罐里的钱可以用在团队活动的时候用。我们会在游戏之夜，用这些钱来买汉堡:o)

这种做法效果不错。不过只有人们常常迟到的团队才需要搞这种制度。有些团队就根本不需要。

火龙果·整理
uml.org.cn

处理“我不知道今天干什么”的情况

有时候，有人会说“我昨天干了这个、这个和这个，但是今天根本不知道该干什么。”这种情况并不少见。那该怎么办呢？

让我们假设 Joe 和 Lisa 两个人都不知道今天该干什么。

如果我是他们的 Scrum master，我会让大家继续讲下去，只是先标记一下哪些人没有事情做。所有人都讲完以后，我会跟团队一起从上到下遍历任务板，检查是否所有条目都已同步，保证所有人都清楚每个条目的含义等等。然后请人添加更多的即时贴上去。接下来我就会对觉得自己没事可干的人说，“我们已经过了一遍任务板，你们现在对今天要做的事情有想法了么？”。希望他们有点儿概念了。

如果他们还不知道该干什么，我会考虑他们是不是可以去跟其他人结对编程。假设 Niklas 今天要实现后台用户管理的 GUI，那我就会有礼貌地建议 Joe 或者 Lisa 去跟 Niklas 结对。这通常都可以有所成效。

要是还不行，那我就会像下面这样干。

Scrum master：“好，下面谁来给我们演示一下这个 beta 版的发布？”（假定这就是当前的 sprint 目标）
Team：疑惑不已，保持沉默。
Scrum master：“我们还没完成？”
Team：“嗯……没有。”
Scrum master：“我靠。为啥还没干完？还剩些什么？”
Team：“我们到现在为止还没有一台测试服务器，构建脚本也出了问题。”
Scrum master：“啊哈！”（向墙上增加了两张贴纸）“Joe 和 Lisa，你们今天能帮我们做点什么呢？”
Joe：“嗯……我可以试着四处找找测试服务器。”
Lisa：“……我可以试着修复构建脚本。”

火龙果 · 整理
uml.org.cn

如果你很幸运的话，有些人会出来把 beta 发布版演示给你看。那就太好了！你已经达成了 sprint 目标。但是如果这时 sprint 只进行了一半呢？很简单。先就已完成的工作向他们表示一下祝贺，然后从任务板右下角的“Next”区域中拿出一两个故事，放到左边的“not checked out”列中。接下来重新进行每日例会。告诉产品负责人一声，你已经把一些条目加进了 sprint。

但是如果团队还没有达成目标，而且 Joe 和 Lisa 还就是不肯做些有用的工作，那又该怎么办？一般我会尝试下面的某种策略（这些都不怎么让人愉快，但已经是无可奈何之计了）：

- **羞辱式做法：** “如果你不知道怎么帮助团队，我建议你还是回家去，或者看书，或者怎么都行。要不也可以找个地方坐下，等别人需要帮忙的时候你就过去。”
- **守旧式做法：**简单给他们分配个任务了事。
- **施加同事压力的做法：** 对他们说，“Joe，还有 Lisa，你们两个可以放松点，我们会站在这里慢慢等，直到你们找到帮助我们完成目标的事情为止。”
- **奴役式做法：**对他们说，“你们今天可以给大伙儿干干杂活。倒咖啡、做按摩、清理垃圾、做午饭，一切一切大家今天让你们做的事情。”你会惊讶的发现 Joe 和 Lisa 在霎那之间就找出了有用的技术任务:o)

如果一个人常常逼得你要这样做，那就应该考虑是不是把他单独找出来做辅导。倘若问题依然存在，你就需要衡量一下这个人对于团队的重要性。

如果他**不是**太重要，就试着把他从你的团队中挪走。

如果他**确实**重要，就试着让他跟别人结对，另一个人充当他的“牧羊人”。Joe 也许是一个优秀的开发人员和架构师，但是他需要别人告诉他应该做什么。没问题。让 Niklas 去做 Joe 永远的牧者。或者你自己承担这个责任。如果 Joe 在你们团队中的作用足够大，那这份投入就是值得的。我们曾有过类似的案例，多少都收到了效果。

火龙果 · 整理
uml.org.cn

9

我们怎样进行 sprint 演示

Sprint 演示（有人也叫它 sprint 回顾）是 Scrum 中很重要的一环，却常为人们低估。

“哦，我们真的**必须**做演示么？没啥好东西能展示的！”
“我们没时间准备&%$#的演示！”
“我没时间参加其他团队的演示！”

为什么我们坚持所有的 sprint 都结束于演示

一次做得不错的演示，即使看上去很一般，也会带来深远影响。

- 团队的成果得到认可。他们会**感觉很好**。
- 其他人可以了解你的团队在做些什么。
- 演示可以吸引相关干系人的注意，并得到重要反馈。
- 演示是（或者说应该是）一种社会活动，不同的团队可以在这里相互交流，讨论各自的工作。这很有意义。
- 做演示会迫使团队**真正完成一些工作**，进行发布（即使是只在测试环境中）。如果没有演示，我们就会总是得到些99%完成的工作。有了演示以后，也许我们完成的事情会变少，但它们是**真正完成**的。这（在我们的案例中）比得到一堆**貌似完成**的工作要好得多，而且后者还会污染下一个 sprint。

如果一个团队或多或少是被逼着做演示的，尤其是他们实际没有完成多少工作的状况下，演示就会变得令人尴尬。团队在做演示的时候会结结巴巴，之后的掌声也会显得勉勉强强。有人会为团队感到

火龙果 · 整理
uml.org.cn

有点儿难过，也有人感到很不爽，因为他觉得宝贵时间被浪费在了一场很烂的演示上。

这会伤害一些人。但它是苦口良药。等到下一个 sprint，这个团队就会真得试着**做完**一些事情！他们会想：“也许我们下个 sprint 可以只演示 2 个功能，而不是 5 个。但这次这些该死的功能一定会正常**工作**！”团队知道这次无论如何他们也要进行演示，一些真正有用的东西被演示出来的机会就会大很多。这种情况我已经目睹很多次了。

Sprint 演示检查列表

- 确保清晰阐述了 sprint 目标。如果在演示上有些人对产品一无所知，那就花上几分钟来进行描述。
- 不要花太多时间准备演示，尤其是不要做花里胡哨的演讲。把那些玩意儿扔一边去，集中精力演示可以实际工作的代码。
- 节奏要快，也就是说要把准备的精力放在保持演示的快节奏上，而不是让它看上去好看。
- 让演示关注于业务层次，不要管技术细节。注意力放在“我们做了什么”，而不是“我们怎么做的”。
- 可能的话，让观众自己试一下产品。
- 不要演示一大堆细碎的 bug 修复和微不足道的特性。你可以提到一些，但是不要演示，因为它们通常会花很长时间，而且会分散大家的注意力，让他们不能关注更加重要的故事。

火龙果 · 整理
uml.org.cn

处理“无法演示”的工作

团队成员：“我不打算演示这个条目，因为它没法被演示。这个故事是‘提高系统的可扩展性，能够容纳 10000 个用户的并发请求’。我豁出命去也没法邀请 10000 个用户同时来做演示，不是吗？”

Scrum master：“那你做完了吗？”

团队成员：“当然。”

Scrum master：“你怎么知道呢？”

团队成员：“我在性能测试环境中搭好了系统，启动 8 个负载服务器，用并发请求做了测试。”

Scrum master：“但是你有没有迹象可以表明系统能够处理 10000 个用户呢？”

团队成员：“是的。测试机挺烂的，不过在测试时还是能处理 50000 个并发请求。”

Scrum master：“你怎么知道的？”

团队成员（被折磨的要抓狂）：“我有报告啊！你可以自己看，报告上都有怎么配置测试环境，发出了多少个请求！”

Scrum master：“那太好了！那就是你的‘演示’啊。给大家看看你的报告就行了。这比什么都没有强，不是吗？”

团队成员：“哦？这就够了吗？不过报告挺难看的，得花点时间美化一下。”

Scrum master：“好的，不过不要花太多时间。不用很好看，只要能传递信息就行。”

火龙果 · 整理
uml.org.cn

10

我们怎样做 sprint 回顾

为什么我们坚持所有的团队都要做回顾

在有关回顾的种种一切中，最重要的就是**确保回顾能够进行**。

由于某些原因，团队常常都不太愿意做回顾。如果不给他们点温柔的刺激，我们的大多数团队都会跳过回顾，直接进行下一个 sprint。也许这只是瑞典的文化，我不太确定。

不过，看起来每个人都觉得回顾的用途极大。说句实话，我认为回顾是 Scrum 中第二重要的事件（最重要的是 sprint 计划会议），因为这是你**做改进的最佳时机**！

当然，你不需要在回顾会议上得到什么好点子，在家中的浴盆里就能做得到！但是团队会接受你的想法么？也许吧，不过如果某个主意是“来自团队”，换句话说，在回顾会议上，每个人都可以贡献和讨论想法，这时候得到某个主意，它会更容易被大家接受。

如果没有回顾，你就会发现团队在不断重犯同样的错误。

火龙果 · 整理
uml.org.cn

我们如何组织回顾

根据情况不同，我们常用的做法也会有些差异，但是一般都会做以下这些事情：

- 根据要讨论的内容范围，设定时间为 1 至 3 个小时。
- 参与者：产品负责人，整个团队还有我自己。
- 我们换到一个封闭的房间中，或者舒适的沙发角，或者屋顶平台等等类似的场所。只要能够在不受干扰的情况下讨论就好。
- 我们一般不会在团队房间中进行回顾，因为这往往会分散大家的注意力。
- 指定某人当秘书。
- Scrum master 向大家展示 sprint backlog，在团队的帮助下对 sprint 做总结。包括重要事件和决策等。
- 我们会轮流发言。每个人都有机会在不被人打断的情况下讲出自己的想法，他认为什么是好的，哪些可以做的更好，哪些需要在下个 sprint 中改变。
- 我们对预估生产率和实际生产率进行比较。如果差异比较大的话，我们会分析原因。
- 快结束的时候，Scrum master 对具体建议进行总结，得出下个 sprint 需要改进的地方。

我们的回顾会议一般没有太规整的结构。不过潜在的主题都是一样的：“我们怎样才能在下个 sprint 中做的更好”。

火龙果 · 整理
uml.org.cn

这是我们近期一次回顾的白板：

图中的三列分别是：

- **Good**：如果我们可以重做同一个 sprint，哪些做法可以保持。
- **Could have done better**：如果我们可以重做同一个 sprint，哪些做法需要改变。
- **Improvements**：有关将来如何改进的具体想法。

第一列和第二列是回顾过去，第三列是展望将来。

团队通过头脑风暴得出所有的想法，写在即时贴上，然后用“圆点投票”来决定下一个 sprint 会着重进行哪些改进。每个人都有三块小磁铁，投票决定下个 sprint 所要采取措施的优先级。他们可以随意投票，也可以把全部三票投在一件事情上。

根据投票情况，他们选出了 5 项要重点进行的过程改进，在下一个回顾中，他们会跟踪这些改进的执行情况。

不过不要想一口吃成个胖子，这一点很重要。每个 sprint 只关注几个改进就够了。

火龙果 · 整理
uml.org.cn

在团队间传播经验

一般来说，在 sprint 回顾中得出的信息都特别有价值。团队之所以很难全心投入工作，是不是因为销售经理常常揪出开发人员去在销售会议上充当“技术专家”？这条信息很重要。或许其他团队也有相同问题？我们是不是应该把更多的产品知识教给产品管理人员，让他们能自己做销售支持？

Sprint 回顾不只关注团队怎样才能在下个 sprint 中做得更好，它有更广袤的含义。

我们的处理策略比较简单。有一个人（我们这儿是我）会参加所有的 sprint 回顾会议，充当知识桥梁。不用太正二八经。

另一种方式，是让每个 Scrum 团队都发布 sprint 回顾报告。我们试过这么做，但发现很多人都不会去读报告，而就此展开改进的就更少了。所以我们还是用了上面那种简单的方式。

充当“知识桥梁“的人需要服从一些重要规则：

- 他应当是一个很好的倾听者。
- 如果回顾会议过于沉寂，他应该问一些简单而目标明确的问题，以刺激团队展开讨论。例如“如果时间可以倒流，从第一天重新开始这个 sprint，那你觉得哪些事情会用其它方式来做？”
- 他应该自愿花时间参加所有团队的全部回顾。
- 他应该有一定的行政权力，如果出现一些团队无法控制的改进建议，他可以帮助推进实施。

这种做法确实很棒，不过也许还有其他更好的方式，如果你知道的话，还请指点我一下。

变，还是不变

假设团队总结出的结论是：“我们团队内部交流的太少了， 所以总是会重复彼此做过的工作，而且把其他人的设计搞得一团糟。”

火龙果 · 整理
uml.org.cn

我们该怎么做呢？引入每日设计会议？引入有助于交流的新工具？增加更多的 wiki 页面？唔，也许吧。不过也不一定。

我们发现：很多时候，只要能清楚地指出问题所在，到了下一个 sprint，问题也许就自行解决了。把 sprint 回顾结果贴在团队房间的墙上（我们常常忘了这一点，可真丢人！）会更有效。在 sprint 中引入的每一点变化，都会让团队付出相应的代价；在引入变化之前，可以先考虑什么都别做，寄希望于问题自动消失（或变小）。

上面（“我们团队内部交流的太少了……”）就是一个很典型的例子，说明什么都不做就有可能解决问题。

如果每次有人发几句牢骚，你就引入新的变化，那人们就不愿意再说小问题了，这就大为不妙。

回顾中发现的问题示例

下面是在 sprint 回顾会议上常常会发现的一些问题，以及相应的典型处理动作。

“我们应花更多时间，把故事拆分成更小的条目和任务”

这个问题很普遍。每天的例会上，都会有人说“我真的不知道今天该干什么”。所以在每一个例会之后，你都要花些时间来找出具体任务。通常这些事情提前做会更有效率。

典型动作：无。团队很可能会在下一个 sprint 计划会议上自己解决掉这个问题。如果它重复出现的话，就延长 sprint 计划会议的时间。

“太多的外界干扰”

典型动作：

- 让团队在下一个 sprint 上减少投入程度，这样就可以有更合理的计划。
- 让团队在下一个 sprint 上把干扰因素记录得更清楚一些：谁带来的干扰，占用了多长时间。也许这可以帮助我们在下次更好地解决问题。

火龙果 · 整理
uml.org.cn

- 让团队试着将所有的干扰因素转给 scrum master 或产品负责人。
- 让团队指定一个人充当“守门员”，所有的干扰都要经由他处理，其他人就可以把注意力保持在项目上。扮演者可以是 Scrum master，也可以大家轮流。

“我们做出了过度的承诺，最后只完成了一半工作”
典型动作：无。下一次这个团队就不会过度承诺了，或者至少不会像这次一样承诺得这么多。

“我们办公室的环境太吵太混乱了”
典型动作：

- 试着创建一个更好的环境，或者把团队搬出去。租一间宾馆的房间。怎样都行。参见“我们怎样布置团队房间”。
- 如果不可能的话，那就让团队在下次 sprint 上降低投入程度，并明确注明这是由于嘈杂混乱的环境导致的。希望这可以让产品负责人开始找上层管理者反映这种问题。

幸运的是，我遇到的状况还没有糟糕到要威胁把团队搬出办公室去。如果被逼无奈的话，我会这样做的:o)

火龙果 · 整理
uml.org.cn

11

Sprints 之间的休整时刻

在实际生活中，你不可能一直像上紧了发条一样始终高速工作。你需要在冲刺的间歇休息。如果弦总是绷得那么紧，实际上收到的成效反而不好。

这对 Scrum 和软件开发也一样。Sprints 安排得很紧凑。作为开发人员，你不会有偷懒的机会，每天你都得在那个该死的会议上站起来告诉每个人你昨天完成了什么。几乎没人愿意说："我昨天基本上一直把腿翘在桌子上，看博客，喝卡布基诺。"

除了真正的休息以外，还有一个很好的理由让我们在 sprints 之间进行修整。Sprint 演示和回顾结束以后，团队和产品负责人都有一大堆信息和想法需要消化。如果他们立刻计划下一个 sprint，那就没人能有机会消化现有的信息或是学到的经验，产品负责人也没有时间在 sprint 演示以后调整优先级。

比较差的安排：

Monday

09-10: Sprint 1 demo
10-11: Sprint 1 retrospective

13-16: Sprint 2 planning

火龙果 · 整理
uml.org.cn

我们试着在启动新的 sprint 之前先进行某种形式的修整（精确地说，是在 sprint 回顾**之后**，下一个 sprint 计划会议**之前**）。不过我们也失败过。

但最起码，我们会力求保证不在同一天举行 sprint 回顾和下一个 sprint 计划会议。在启动新的 sprint 之前，每个人都应该至少度过一个不需要考虑 sprint 的夜晚。

好一些：

Monday	Tuesday
09-10: Sprint 1 demo 10-11: Sprint 1 retrospective	9-13: Sprint 2 planning

更好：

Friday	Saturday	Sunday	Monday
09-10: Sprint 1 demo 10-11: Sprint 1 retrospective			9-13: Sprint 2 planning

“实验日”（你爱叫什么都行）算是一种方式。在这样的日子里，开发人员基本上可以做任何他想做的事情（OK，我承认这种想法是从 Google 来的）。比如研习最新的工具和 API、准备认证、跟同事讨论乱七八糟的事情、开发自己喜欢的项目，等等。

我们的目标是在每个 sprint 之间安排一个实验日。这样你就能得到自然的休息，开发团队也能让自己了解最前沿的知识。这也是一种能够吸引员工的福利。

最好?

Thursday	Friday	Saturday	Sunday	Monday
09-10: Sprint 1 demo 10-11: Sprint 1 retrospective	LAB DAY			9-13: Sprint 2 planning

火龙果 · 整理
uml.org.cn

目前我们每个月有一次实验日，放在每月的第一个星期五。为什么不放在 sprints 之间呢？唔，因为我觉得整个公司应当在同样的时间度过实验日，否则就会有人不上心。而且我们（到目前为止）还没有把所有产品的 sprint 时间安排都协调一致，所以我不得不选一个跟 sprint 无关的实验日。

也许有一天我们会试着对所有产品的 sprint 进行同步（也就是所有的产品跟团队都有相同的 sprint 启动时间和结束时间）。这时候我们就肯定会选择两个 sprint 之间的日子当实验日了。

火龙果 · 整理
uml.org.cn

12

怎样制定发布计划，处理固定价格的合同

有时候，一次只计划一个 sprint 中要做的事情会略显不足，我们还得提前多做些计划。尤其是签了固定价格的合同之后，我们就**不得不**预先计划了，不然就会有无法按期交付的危险。

一般来讲，制定发布计划是在尝试回答这个问题：“**最晚到什么时候为止，**我们可以交付这个新系统的 1.0 版本？”

如果你**真的**想学习有关发布计划的知识，我建议你还是跳过这章，去买本 Mike Cohn 的书《敏捷估计与规划》。我真希望能够早点读到这本书（我是在自己解决完这种问题**之后**才读到它的……） 我对发布计划的认识比较简单，不过用来入门也差不多了。

定义你的验收标准

除了普通的产品 backlog 之外，产品负责人还会定义一系列的**验收标准**，它从合同的角度将产品 backlog 中重要性级别的含义进行了简单分类。

下面是验收标准规则的一个例子：

- 所有重要性>= 100 的条目都**必须**在 1.0 版中发布，不然我们就会被罚款到死翘翘。
- 所有重要性在 50 - 99 之间的条目**应该**在 1.0 中发布，不过**也许**我们可以在紧接着的一个快速发布版中完成这些。
- 重要性在 25 – 49 之间的条目也都是需要的，不过可以在 1.1 版中发布。

火龙果 · 整理
uml.org.cn

- 重要性< 25 的条目都是不确定的，也许永远不会用到。

下面是一个产品 backlog 的例子，根据上面的规则标记了不同颜色。

重要性	名称
130	banana
120	apple
115	orange
110	guava
100	pear
95	raisin
80	peanut
70	donut
60	onion
40	grapefruit
35	papaya
10	blueberry
10	peach

红= **必须**在 1.0 版中发布（banana – pear）
黄= **应该**在 1.0 版中发布（raisin – onion）
绿= 也许可以以后再做（grapefruit – peach）

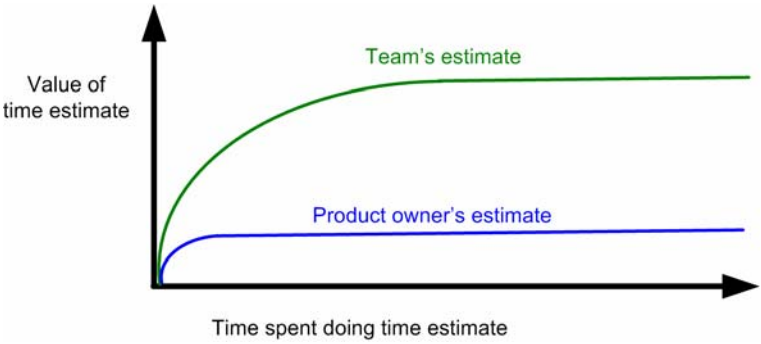
所以如果我们在最后期限之前能够发布从 banana 到 onion 的所有条目，我们就是安全的。如果时间不够用的话，也许我们可以跳过 raisin、peanut、donut、onion。Onion 以下的东西都算是额外的了。

对最重要的条目进行时间估算

为了制定发布计划，产品负责人需要进行时间估算，至少是要估算在合同中包含的故事。跟 sprint 计划会议一样，这是产品负责人和团队协作共同完成的——团队进行估算，产品负责人描述条目内容，回答问题。

火龙果 · 整理
uml.org.cn

如果时间估算最后被证明接近正确结果，那它就是**有价值**的；如果结果有所偏离，例如偏差了 30%，价值则有所降低；如果它跟实际结果一点关系都没有，那就完全没用了。
下面是我根据做估算的人、做估算所用时间以及估算的价值三者之间的关系所画的一张图。



把图中的含义换成文字来表述就显得有些罗嗦：

- 让**团队**来做估算。
- 不要让他们花太多时间。
- 确保他们理解时间估算只是**粗略估算**，而不是**承诺**。

通常产品负责人会把整个团队聚到一个房间，提供一些食品饮料，告诉他们这个会议的目标是得出产品 backlog 上前 20 个（或多少都行）故事的时间估算。他先讲一遍所有的故事，然后让团队开始工作。他会待在房间里，回答大家的问题，必要时解释清楚每一个条目的内容范围。就像做 sprint 计划一样，“如何做演示”这个字段也有助于减少发生误解的风险。

这个会议的时间必须要严格限制，不然团队就会把大量时间花费在少数几个故事上。

如果产品负责人想在这上面花更多的时间，他可以随后再安排一个会议。团队必须保证产品负责人可以清楚地认识到这些会议对他们

火龙果 · 整理
uml.org.cn

当前 sprint 的影响，这样他就能理解时间估算这个活动本身也是有代价的了。

下面是一个时间估算结果的例子（以故事点表示）：

重要性	名称	估算
130	banana	12
120	apple	9
115	orange	20
110	guava	8
100	pear	20
95	raisin	12
80	peanut	10
70	donut	8
60	onion	10
40	grapefruit	14
35	papaya	4
10	blueberry	
10	peach	

估算生产率

OK，现在我们对最重要的故事有了一些粗略的时间估算。下一步是估算每个sprint的平均生产率。

这就意味着我们要确定我们的投入程度。请参见“团队如何决定哪些故事放到sprint中”。

投入程度表示“团队有多少时间可以放在当前所承诺的故事上”。它永远不可能是100%，因为团队会把时间用于完成未经计划的条目、切换环境、帮助其他团队、检查邮件、修复自己出问题的电脑、在厨房中讨论政治等等。

假设我们决定了团队的投入程度是50%（相当低了，一般我们都是70%左右），sprint长度是3个星期（15天），团队是6个人。

火龙果 · 整理
uml.org.cn

这样来看每个 sprint 都是 90 个人- 天，但是只能完整交付 45 个人-天的故事（投入程度是 50%）。

所以我们的估算生产率是45个故事点。

如果每个故事的估算都是5天（实际不是），那团队差不多就能在一个sprint中完成9个故事。

统计一切因素，生成发布计划

现在我们有了时间估算和生产率（45），可以很容易的把产品 backlog 拆到 sprints 中：

重要性	名称	估算
		Sprint 1
130	banana	12
120	apple	9
115	orange	20
		Sprint 2
110	guava	8
100	pear	20
95	raisin	12
		Sprint 3
80	peanut	10
70	donut	8
60	onion	10
40	grapefruit	14
		Sprint 4
35	papaya	4
10	blueberry	
10	peach	

火龙果 · 整理
uml.org.cn

在不超出 45 这个估算生产率的前提下，我们把每个 sprint 都尽可能塞满了故事。
现在我们知道大约需要 3 个 sprint 来完成所有“必须要的”和“应该要的”。

3 sprints = 9 个星期 = 2 个月。这是我们要向客户许诺的最后期限么？要视合同情况，范围限制有多严格，等等而定。我们通常都会增加相当多的时间缓冲，以避免糟糕的时间估算、未预期的问题和未预期的特性等造成影响。在这种情况下，我们可能会同意把发布日期定在三个月后，让我们“保留”一个月。

我们可以每隔三个星期就给客户演示一些有用的东西，并在过程中邀请他们更改需求（当然也要看是什么样的合同），这很不错。

调整发布计划

现实不会调整自己来适应计划，所以我们必须走另外一条路。

每个 sprint 之后，我们都要看一下这个 sprint 的实际生产率。如果实际生产率跟估算生产率差距很大，我们就会给下面的 sprint 调整生产率，更新发布计划。如果这会给我们带来麻烦，产品负责人就会跟客户进行谈判；或者检查一下是否能够在不违反合同的情况下调整范围；或者他跟团队一起找出一些方法，通过消除某些在 sprint 中发现的严重障碍，提高生产率或是投入程度。

产品负责人也许会给客户打电话说，“嗨，我们目前比进度稍微慢了点，不过我相信如果把‘embedded Pacman’这个特性去掉的话，我们就可以在期限之前完工，因为构建它会用我们很多时间。如果你同意的话，我们可以在第一次发布后三周内的后续发布中把它加进去。”

可能这对客户来说不是好消息，但至少我们是诚实的，并且尽早给客户提供了选择——我们是应该准时发布最重要的功能，还是推延一段时间，发布所有的功能。做出这种选择通常都不是难事:o)

火龙果 · 整理
uml.org.cn

13

我们怎样组合使用 Scrum 和 XP

要说组合使用 Scrum 和 XP（极限编程）可以带来累累硕果，这毫无争议。我在网上看到过的绝大多数资料都证实了这一点，所以我不会花时间去讨论为什么要这么做。

不过，我还是会提到一点。Scrum 注重的是管理和组织实践，而 XP 关注的是实际的编程实践。这就是为什么它们可以很好地协同工作——它们解决的是不同领域的问题，可以互为补充，相得益彰。

所以，我在这里要向现有的实践证据中加上我自己的声音：组合使用 Scrum 和 XP 会有显著收获！

下面我会着重讲述 XP 中最有价值的一些实践，以及我们在每日工作中的应用方式。我们的团队并没有全都把所有的实践都试过一遍，但总的来说，在绝大多数层面上组合使用 XP 与 Scrum，我们都已经尝试过了。有些 XP 实践直接被 Scrum 解决掉了，可以被视作二者的重叠。如“整体团队”，“坐在一起”，“故事”和“计划游戏”。在这些情况下我们就直接使用了 Scrum。

结对编程

我们近来开始在一个团队中实施结对编程。效果相当好。虽然其他团队大多数还没有进行太多尝试，但在一个团队中使用了几个 sprint 之后，我已经有了很高的热情去指导其他团队进行试用。

下面是到目前为止有关结对编程的一些结论：

火龙果 · 整理
uml.org.cn

- 结对编程可以提高代码质量。
- 结对编程可以让团队的精力更加集中（比如坐在你后面的那个人会提醒你，“嘿，这个东西真的是这个 sprint 必需的吗？”）。
- 令人惊奇的是，很多强烈抵制结对编程的开发人员根本就没有尝试过，而一旦尝试之后就会迅速喜欢上它。
- 结对编程令人精疲力竭，不能全天都这样做。
- 常常更换结对是有好处的。
- 结对编程可以增进团队间的知识传播。速度快到令人难以想象。
- 有些人就是不习惯结对编程。不要因为一个优秀的开发人员不习惯结对编程就把他置之不理。
- 可以把代码审查作为结对编程的替代方案。
- “领航员”（不用键盘的家伙）应该自己也有一台机器。不是用来开发，而是在需要的时候稍稍做一些探索尝试、当“司机”（使用键盘的家伙）、遇到难题的时候查看文档，等等。
- 不要强制大家使用结对编程。鼓励他们，提供合适的工具，让他们按照自己的节奏去尝试。

测试驱动开发（TDD）

阿门！对我来说，它比 Scrum 和 XP 还要重要。你可以拿走我的房子、我的电视还有我的狗，但不要试着让我停止使用 TDD！如果你不喜欢 TDD，那就别让我进入你的地盘，不然我一定会想方设法来偷摸着干的 :o)

下面是有关 TDD 的一个 10 秒钟总结：

> **测试驱动开发意味着你要先写一个自动测试，然后编写恰好够用的代码，让它通过这个测试，接着对代码进行重构，主要是提高它的可读性和消除重复。整理一下，然后继续。**

人们对测试驱动开发有着各种看法：

火龙果 · 整理
uml.org.cn

- TDD 很难。开发人员需要花上一定时间才能掌握。实际上，往往问题并不在于你用了多少精力去教学、辅导和演示——多数情况下，开发人员掌握它的唯一方式就是跟一个熟悉 TDD 的人一起结对编程，一旦掌握以后，他就会受到彻底的影响，从此再也不想使用其它方式工作。
- TDD 对系统设计的正面影响特别大。
- 在新产品中，需要过上一段时间，TDD 才能开始应用并有效运行，尤其是黑盒集成测试。但是回报来得非常快。
- 投入足够的时间，来保证大家可以很容易地编写测试。这意味着要有合适的工具、有经验的人、提供合适的工具类或基类，等等。

我们在测试驱动开发中使用了如下工具：

- jUnit / httpUnit / jWebUnit。我们正在考虑使用 TestNG 和 Selenium.
- HSQLDB 用作嵌入式的内存数据库，在测试中使用。
- Jetty 用作嵌入式的内存 Web 容器，在测试中使用。
- Cobertura 用来度量测试覆盖率。
- Spring 框架用来织入不同类型的测试装置（带有 mock、不带 mock、带有外部数据库或带有内存数据库等等）。

在我们那些经验最丰富的产品中（从 TDD 的视角来看），都有自动化的黑盒验收测试。这些测试会在内存中启动整个系统，包括数据库和 Web 服务器，然后只通过系统的公共接口进行访问（例如 HTTP）。

它会把开发-构建-测试这三者构成的循环变得奇快无比，同时还可以充当一张安全网，让开发人员有足够的信心频繁重构，伴随着系统的增长，设计依然可以保持整洁和简单。

在新代码上进行 TDD

我们在所有的全新开发过程中都使用 TDD，即便这会在开始时延长项目配置时间（因为我们需要更多的工具，并为测试装备提供支

火龙果 · 整理
uml.org.cn

持等等）。其实用脚指头思考也可以知道，TDD 带来的好处如此之大，还有什么理由可以不用它呢。

在旧代码上进行 TDD

TDD 是很难，但是在一开始没有用 TDD 进行构建的代码库上实施 TDD……则是**难上加难**！为什么？嗯，实际上，就这个话题我可以写上许多页，所以我想最好到此为止。也许我会在我的下一个论文“硝烟中的 TDD”中进行解释:o)

我们曾花了大量的时间，在一个比较复杂的系统上进行自动化集成测试，它的代码库已经存在很长时间了，处于极度混乱的状态，一丁点的测试代码都没有。

每次发布之前，都有一个由专门的测试人员组成的团队，来进行大批量的、复杂的回归测试和性能测试。那些回归测试大多数都是手工进行。我们的开发和发布周期就这样被严重延误了。我们的目标是将这些测试自动化，但是几个月的痛苦煎熬以后，仍然没有取得多少进展。

之后我们改变了方式。首先承认自己已经陷入了手工回归测试的泥潭，然后再来问自己：“怎么让手工回归测试消耗的时间更少呢？”当时开发的是一个赌博系统，我们意识到：测试团队在非常琐碎的配置任务上花费了大量的时间。例如浏览后台并创建牌局来测试，或者等待一个安排好的牌局启动。所以我们特地创建了一些实用工具。这些快捷方式和脚本很小，而且使用方便。它们可以完成那些乱七八糟的工作，让测试人员专注真正的测试。

这些付出确实收到了成效！实际上，我们的确应该从一开始就这样做。当初太急于将测试自动化了，都忘了应该一步一步走。刚开始应该想办法提高**手工**测试的效率。

学到的一课：如果你深陷手工回归测试的泥潭，打算让它自动化执行，最好还是放弃吧（除非做起来特别简单）。首先还是应该想办法简化手工回归测试。**然后**再考虑将真正的测试变成自动化执行。

火龙果 · 整理
uml.org.cn

增量设计

这表示一开始就应该保持设计简单化，然后不断进行改进；而不是一开始努力保证它的正确性，然后就冻结它，不再改变。

在这一点上我们做的相当好，我们用了大量的时间来做重构，改进既有设计，而几乎没用什么时间来做大量的预先设计。有时候我们当然也会出错，例如允许一个不稳定的设计“陷入”太深，以至于后来代码重构成了一个大问题。不过总体来看我们都是相当满意的。

持续的设计改进，这在很大程度上是 TDD 自动带来的成果。

持续集成

我们的大多数产品在持续集成方面都已经很成熟了，它们是基于 Maven 和 QuickBuild 的。这样做很管用，而且节省了我们大量时间。对于 “哎，它在**我的**电脑上没有问题”这样的老问题，持续集成也是终极解决方案。要判断所有代码库的健康状况，可以用持续构建服务器充当“法官”或是参考点。每次有人向版本控制系统中 check in 东西，持续构建服务器就会醒来，在一个共享服务器上从头构建一切，运行所有测试。如果出现问题，它就会向整个团队发送邮件告知大家构建失败，在邮件中会包括有哪些代码的变化导致构建失败的精确细节，指向测试报告的链接等。

每天晚上，持续构建服务器都会从头构建产品，并且向我们的内部文档门户上发布二进制文件（ears，wars 等）、文档、测试报告、测试覆盖率报告和依赖性分析报告等等。有些产品也会被自动部署到测试环境中。

把这一切搭建起来需要**大量工作**，但付出的每一分钟都物有所值。

火龙果 · 整理
uml.org.cn

代码集体所有权

我们鼓励代码集体所有权，但并不是所有团队都采取了这种方式。我们发现：在结对编程中频繁交换结对，会自动把代码集体所有权提到一个很高的级别。我们已经证实，如果团队拥有高度的代码集体所有权，这个团队就会非常健壮，比如某些关键人物生病了，当前的 sprint 也不会因此嗝屁朝凉。

充满信息的工作空间

所有团队都可以有效利用白板和空的墙壁空间。很多房间的墙上都贴满了各种各样关于产品和项目的信息。这样做最大的问题，就是那些旧的作废信息也堆在墙上，也许我们应该在每个团队中引入一个“管家”的角色。

我们鼓励使用任务板，但是并不是所有团队都采用了它。参见“我们怎样布置团队空间”。

代码标准

不久前我们开始定义代码标准。它的用处很大，要是我们早就这样做就好了。引入代码标准几乎没花多少时间，我们只是一开始从简单入手，让它慢慢增长。只需要写下不是所有人都了如指掌的事情，并尽可能加上对外部资料的链接。

绝大多数程序员都有他们自己特定的编程风格。例如他们如何处理异常，如何注释代码，何时返回 null 等等。有时候这种差异没什么关系，但在某些情况下，系统设计就会因此出现不一致的现象，情况严重，代码也不容易看懂。这时代码标准的用处就会凸显，从造成影响的因素中就可以知道了。

下面是我们代码标准中的一些例子：

- 你可以打破这里的任一规则，不过一定要有个好理由，并且记录下来。

火龙果 · 整理
uml.org.cn

- 默认使用 Sun 的代码惯例：http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html
- 永远，永远，永远不要在没有记录堆栈跟踪信息（stack trace）或是重新抛出异常的情况下捕获异常。用 log.debug() 也不错，只要别丢失堆栈跟踪信息就行。
- 使用基于 setter 方法的依赖注入来将类与类解耦（当然，如果紧耦合可以令人满意的话就另当别论）。
- 避免缩写。为人熟知的缩写则可以，例如 DAO。
- 需要返回 Collections 或者数组的方法不应该返回 null。应该返回空的容器或数组，而不是 null。

可持续的开发速度/精力充沛的工作

很多有关敏捷软件开发的书都声称：加班工作在软件开发中会降低生产率。

经过几次不情愿的试验之后，我完全拥护这种说法！

大约一年以前，我们中有一个团队（最大的团队）在疯狂加班。现存代码库的质量惨不忍睹，他们不得不投入绝大多数的时间来救火。测试团队（同样也在加班）根本没时间来认真地做质量保证工作。我们的用户很生气，小道流言也快把我们活活吞掉了。

几个月后，我们成功地把大家的工作时间缩短到了适当的范围。他们正常上下班（除了有时候在项目关键期要加班以外）。令人惊异的是，生产率和质量都取得了显著提高。

当然，减少工作时长绝不是带来改进的**唯一**因素，但我们都确信它的影响很大。

14

我们怎样做测试

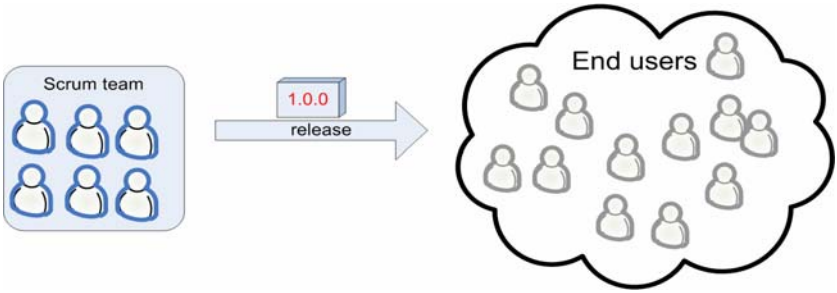
这是最困难的部分。我不知道它到底是只是 Scrum 中最困难的部分，还是在软件开发中通常都是最困难的部分。

在不同组织的各种开发活动中，测试可能是差异最大的。它依赖于你有多少个测试人员、系统类型（只是服务器+web 应用，还是交付完整的软件？）、发布周期的长短、软件的重要性（博客服务器 vs.飞行控制系统），等等。

我们曾经尝试过多种在 Scrum 中做测试的方式。下面我会尽力描述一下我们的做法，以及到目前为止掌握的经验。

你大概没法取消验收测试阶段

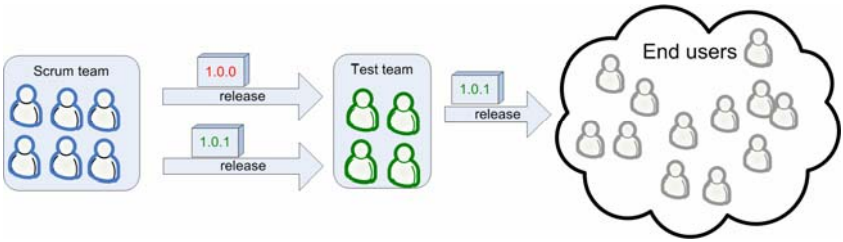
在理想化的 Scrum 世界中，每个 sprint 最终会产生一个可部署的系统版本。那赶紧部署就好了，是么？



不是。

火龙果 · 整理
uml.org.cn

根据我们的经验，这样做一般都是不成的。很恶心的 bug 会因此出现。如果质量对你来说还算重要，你就应该进行验收测试。此时，团队**之外**的专职测试人员会用测试来攻击系统，而且这些测试是 Scrum 团队要么考虑不到，要么没有时间完成，或是限于硬件条件无法完成的。测试人员会采取与终端用户一模一样的方式来操作系统，也就是说他们必须要手工进行测试（假设你的系统用户是人）。



测试团队会发现 bug，Scrum 团队就得发布针对 bug 修复的版本，或早或晚（希望更早一些）你就要为终端用户发布修复了 bug 的 1.0.1 版本，而不是问题重重的 1.0.0 版本。

我说的“验收测试阶段”，是指整个测试、调试、重新发布阶段，直到得到可以用来做产品发布的版本为止。

把验收测试阶段缩到最短

验收测试阶段会让人受不了。那的确让人觉得不太敏捷。虽然我们不能逃避这个阶段，但可以想办法尽量缩短时间。说得更明白一些，把需要花在验收测试阶段上的**时间**减到最少。我们的做法是：

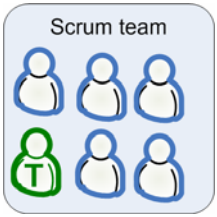
- 全力提高 Scrum 团队交付的代码质量。
- 全力提高人工测试工作的效率（即，找到最好的测试人员；给他们最好的工具；确保他们上报那些耗费时间、却能够被自动化完成的工作）

那我们该怎么提高 Scrum 团队提交的代码质量呢？嗯，办法还是很多的。我们发现下面这两种办法效果很好：

- 把测试人员放到 Scrum 团队中
- 每个 sprint 少做点工作

火龙果 · 整理
uml.org.cn

把测试人员放到 Scrum 团队来提高质量



是的，我听到过对立的意见：

- “很明显啊！Scrum 团队应该是**跨职能**的！”
- “Scrum 团队应该是没角色的！我们不能把**只做**测试的人放到里面来！”

让我澄清一下。这里我说的“测试人员”指的是“主要技能是测试的人”，而不是“只做测试的人”。

开发人员常常都是很差劲的测试人员。**尤其**是他们测试自己代码的时候。

测试人员就是“验收的家伙”

除了“只是”个团队成员以外，测试人员还有个重要的工作要做。他负责验收。Sprint 中的任何工作，如果**他**不说完成，那就不能算完成。我发现开发人员常常说一些工作已经完成了，但事实并非如此。即使你有一个很明确的对“完成”的定义（你确实应该如此，参见“定义‘完成’”），开发人员也会经常忘掉。我们这些编程的人都不怎么有耐心，一心想着尽快去做下一个条目。

那么我们的**测试先生**怎么知道某些事情已经完成呢？嗯，首先，他应该**测试**它！（吃惊吧？）我们经常都会发现：开发人员认为“完成”的工作，却**根本无法测试**！原因包括代码没有提交，或者还没有部署到测试服务器上，等等。一旦测试先生开始测试这个特性，他就应该跟开发人员一起浏览一遍“完成”检查列表（如果你有的话）。例如，如果在“完成”的定义中写着一定要有版本说明，那

火龙果 · 整理
uml.org.cn

测试先生就要去检查是不是有版本说明。如果对这个特性有比较正式的规范说明（我们这里很少有这种情况），测试先生就要据此进行检查。等等不一而足。

妙处由此而生：

这下团队中就有了这样一个人，可以完美地担当组织 sprint 演示的职责。

如果没有任何事情需要测试，那测试人员该做什么？

这个问题会常常出现。测试先生会说："嘿，Scrum master，目前没有什么东西需要测试了，那**我**该做什么呢？"也许团队需要一个星期才能完成第一个故事，那**这段**时候测试人员该做什么呢？

嗯，首先，他应该要**为测试做准备**。包括编写测试规范，准备测试环境等等。开发人员有开发完的功能可供测试以后，就不用再等了，测试先生可以立刻开始测试。

如果团队在做 TDD，从第一天开始，大家都会花时间来编写测试代码，此时测试人员应该跟编写测试代码的开发人员一起结对编程。如果测试人员根本不会编程，他也应该跟开发人员结对，即便他只能坐在一边看，让开发人员敲键盘。相对于好的开发人员，好的测试人员常常能想出多种不同类型的测试，所以他们可以互补。

如果团队没有实施 TDD，或者没有足够的测试用例需要编写，那测试人员可以去随意做一些能够帮助团队达成 sprint 目标的事情。就像其他团队成员一样。如果测试人员会编程，那自然再好不过。如果他不会，你的团队就得找出在 sprint 中需要完成的、而且不用编程的工作。

在 sprint 计划会议中，进行到拆分故事阶段，团队会把注意力放在**编程性任务**上，但一般在 sprint 中都会有很多**非编程性**任务需要完成。如果在 sprint 计划阶段花上一些时间来**找出非编程性任务**，测试先生就有机会来做出大量贡献，即使他不会编程，当前也没有测试工作要做。

火龙果 · 整理
uml.org.cn

下面是在 sprint 中需要完成的非编程性任务的例子：

- 搭建测试环境。
- 明确需求。
- 与运营部门讨论部署的操作细节。
- 编写部署文档（版本说明，RFC，或任何在你们组织中要写的东西）。
- 和外界的资源进行联系（例如 GUI 设计师）。
- 改进构建脚本。
- 将故事进一步拆分成任务。
- 标识出来自开发人员的核心问题，并帮助解决这些问题。

从另一个角度来看，如果测试先生成了瓶颈，那我们该怎么办？假设在 sprint 的最后一天突然完成了很多工作，测试先生根本没有时间测试完所有的事情。我们怎么办？不妨把团队中的所有人都分配给测试先生当助手。他决定哪些事情自己来做，把一些烦人的测试交给团队中的其他人来做。这就是跨职能团队该做的事情！

所以没错，测试先生**确实**在团队中有一个特定的角色，不过他仍然可以做其他工作，其他的团队成员也可以做他的工作。

在每个 sprint 中少做工作来提高质量

回到 sprint 计划会议上。简单来说，就是别把太多故事都放到 sprint 里面去！如果碰到了质量问题，或者验收测试周期太长，干脆就每个 sprint 少干点！这会自动带来质量提升、验收测试周期缩短、影响终端用户的 bug 减少，并在短期内得到更高的生产力，因为团队可以始终关注于新的东西，而不是不断修复出现问题的旧功能。

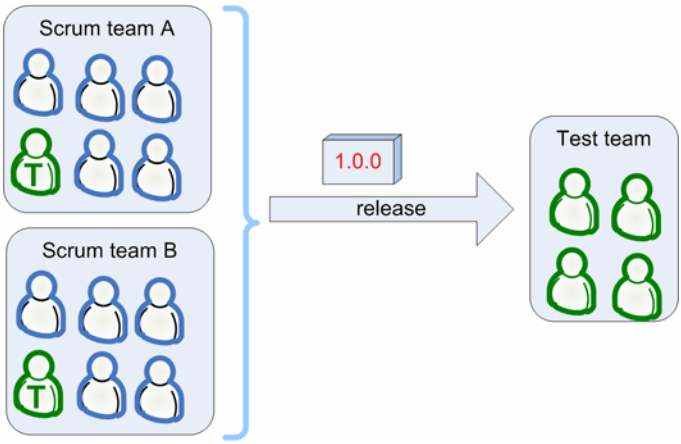
相对于构建大量功能，然后不得不在惊慌失措的状态下做热修复来说，少构建一些功能，但是把它弄得稳定点儿，这样做要合算得多。

验收测试应该作为 sprint 的一部分么？

我们在这里分歧较大。有些团队把验收测试当成了 sprint 的一部分。但大部分团队都没这样做。原因主要有两点：

火龙果 · 整理
uml.org.cn

- Sprint 是有时间盒限制的。验收测试（在我的定义中，它要包括调试和再次发布）的时间却很难固定。如果时间用完了，你还有一个严重的 bug 怎么办？是要带着这个严重 bug 交付上线，还是等到下个 sprint 再说？大多数情况下，这两种解决方案都是不可接受的。所以我们把人工验收测试排除在外。
- 如果有多个团队开发同一个产品，那就得等所有团队的工作成果合并以后，再进行人工验收测试。如果每个团队都在 sprint 中进行人工验收测试，最后还是要有一个团队测试最终版本，而且这个版本集成了全部团队的工作。

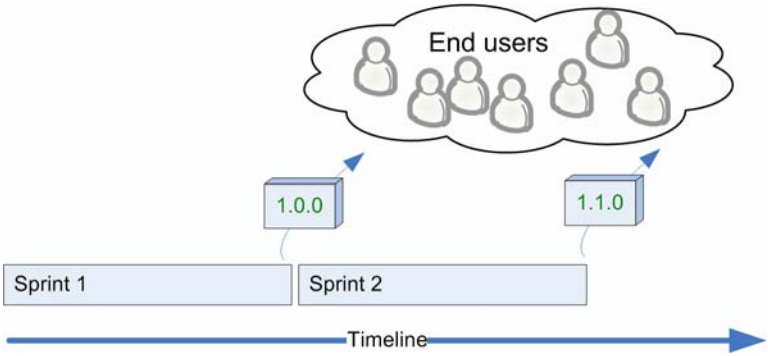


这个解决方案不算完美，但对我们来说，已经可以满足大多数情况的需要了。

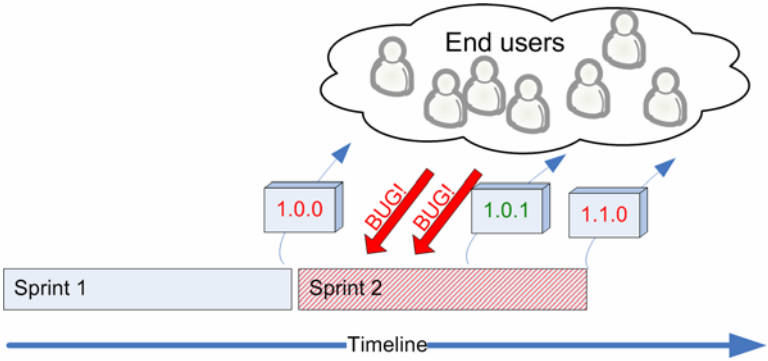
Sprint 周期 vs. 验收测试周期

在完美的 Scrum 世界中，你根本不需要验收测试阶段，因为每个 Scrum 团队在每个 sprint 结束以后，都会发布一个新的可供产品化的版本。

火龙果 · 整理
uml.org.cn



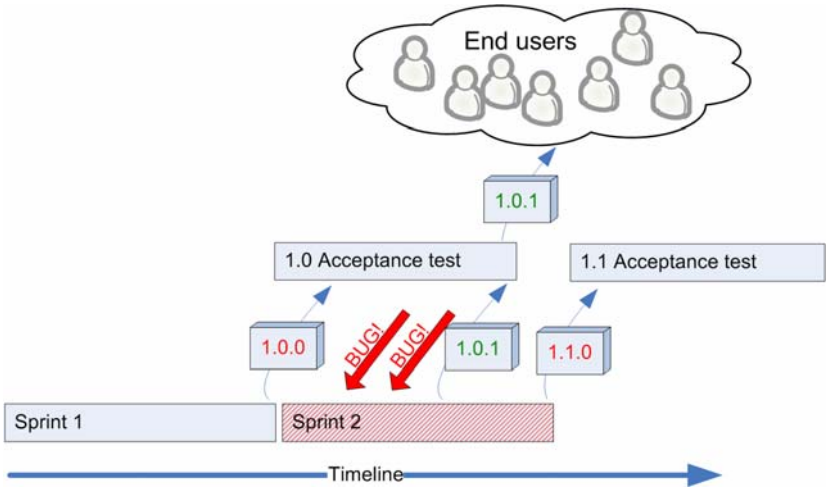
不过，下面这张图就更符合实际情况了：



在 sprint 1 之后，我们得到了满是 bug 的 1.0.0 版本。在 sprint 2 中，bug 报告开始涌入，团队花了大部分的时间来进行调试，然后又被迫在 sprint 的中期发布了修复了 bug 的 1.0.1 版本。到了 sprint 2 末尾，他们发布了 1.1.0 版本，提供了一些新特性，但 bug 数量有增无减，因为他们从上一个版本发布以后就一直被 bug 所干扰，所以能够用来保证代码质量的时间就更少。然后就一直这样循环下去……

在 sprint 2 中的红色斜线表示出了混乱的存在。

不怎么好看是吧？但令人悲哀的是，即使你有验收测试团队，这些问题仍会存在。唯一的区别是，在后者中，大多数 bug 报告会来自于测试团队，而非怒气冲冲的用户。从商业视角来看，二者之间有着很大差别，但对开发人员而言却几乎没什么两样。不过测试人员通常都没有用户那么强势。一般如此。



我们目前还没有发现这个问题的解决方案。不过还是尝试过许多不同的模型。

首先，还是全力提高 Scrum 团队发布的代码质量。在一个 sprint 中及早发现并修复 bug，要比 sprint 结束以后再这样做的代价小得多。

但事实还是事实，就算是我们可以把 bug 数量减少到最小，在 sprint 结束后还是有 bug 报告出来。那我们是怎么做的呢？

方式 1：“在旧版本可以产品化之前，不构建新特性”

听起来挺不错的，不是吗？你是否也有这种温暖舒适的感觉？

我们曾几度差点采用这种方式，而且还画出了想象中如何进行实施的模型。但是意识到它的负面影响后，我们就改变了主意。如果这

火龙果 · 整理
uml.org.cn

样做的话，我们就得在 sprint 之间添加一个无时间限制的发布阶段，而且在这个时期内只能进行测试和调试，直到可以做出产品发布来为止。

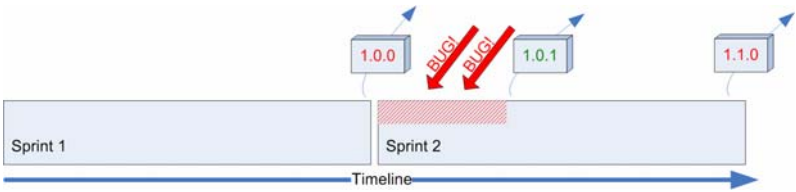
我们不喜欢在 sprints 之间加上无时限的发布阶段，主要是因为它可能会破坏 sprint 的节奏。我们再也无法说出“每三周启动一个新的 sprint”这样的话来。另外，它也没法根除问题。即使有一个发布阶段，依然会不时出现紧急的 bug 报告，我们不得不为它们做好准备。

方式 2：“可以开始构建新东西，但是要给‘将旧功能产品化’分配高优先级”

这是我们最喜欢的方式。至少现在如此。

一般我们完成一个 sprint 以后就会开始进行下一个。但是我们会在接下来的 sprint 中花一些时间解决过往 sprint 中留下的 bug。如果修复 bug 占用了太多时间，从而导致接下来的 sprint 遭到严重破坏，我们就会分析问题产生的原因以及如何提高质量。我们会确保 sprint 的长度，使之足以完成对上个 sprint 中一定数量 bug 的修复。

随着时间推移，经过几个月以后，修复上个 sprint 遗留 bug 所用的时间就会减少。而且当 bug 发生以后，所牵扯的人也更少了，所以不会总是干扰**整个**团队。现在这种做法已经得到了更多人的认可。



火龙果 · 整理
uml.org.cn

在 sprint 计划会议上，考虑到会花时间修复上个 sprint 的 bug，所以我们会把投入程度设得足够低。经过一段时间，团队在估算方面已经做得很到位了。生产率度量也起到了很大帮助作用（参见 24 页，“团队如何决定哪些故事放到 sprint 中？”）。

糟糕的方式——“只关注构建新东西”

它实际的含义是“关注构建新东西，而不是把旧的产品化”。真有人会这么干？嗯，我们刚开始的时候就常犯这样的错误，我也相信很多公司也是如此。这种症状跟压力有关。很多经理都不能真正理解：即使所有的编程活动都已完成，距离产品发布还有很遥远的距离。至少复杂系统是这样的。所以经理（或者产品负责人）要求团队继续增加新特性，而大家手中那些“差不多可以发布”的代码就越来越多，整个工作的速度都会因此而放缓。

别把最慢的一环逼得太紧

假设验收测试是你那里最慢的一环。测试人员稀缺，或者低劣的代码质量造成了过长的验收测试周期。

假设你的验收测试团队每星期最多测试三个特性（不，我们不会用“每周测试的特性”来进行度量，我只是在这个例子中用一下而已）。而开发人员每星期能够开发 6 个特性。

经理或者产品负责人（甚至团队）会觉得不妨安排每周开发 6 个特性。

千万不要！你最终一定会认识到现实的残酷，可那时伤害业已造成。

实际上，应该安排每周只完成 3 个特性，多余的时间用来攻克测试的瓶颈。例如：

- 让一些开发人员去做测试人员的工作（呃，他们会因此而爱你的……）。
- 实现一些工具或脚本，用来简化测试工作。
- 增加更多的自动化测试代码。

火龙果 · 整理
uml.org.cn

- 延长 sprint 长度，把验收测试放到 sprint 里面来。
- 把一些 sprint 定义为“测试 sprint”，其中整个团队都作为验收测试团队进行工作。
- 雇佣更多测试人员（即使这会意味着减少开发人员）。

这些解决方案我们全都试过（除了最后一点）。最好的长期解决方案当然是第 2 和第 3 点，即更好的工具与脚本，还有测试自动化。

回顾可以帮助我们更好地识别出最慢的一环。

回到现实

也许我的话会让你认为：我们在所有的 Scrum 团队中都有测试人员，针对每个产品都有大规模的验收测试团队，在每个 sprint 结束以后都会进行发布，等等等等。

其实，我们也没有做到。

我们**有几次**能成功地做到这种程度，也能看到它所带来的正面影响。但我们的质量保证过程想要得到认可，还有很长的路要走，我们仍有很多东西要学。

火龙果 · 整理
uml.org.cn

15

我们怎样管理多个 Scrum 团队

在多个 Scrum 团队开发同一个产品的状况下，很多事情都会变得更加复杂、棘手。这个问题普遍存在，跟 Scrum 没太大关系。更多开发者=更多复杂情况。

我们也（和往常一样）碰到过这种情况。人最多的时候，曾有大约 40 个人开发同一个产品。

这里的核心问题是：

- 要创建多少个团队
- 如何把人员分配到各个团队中

创建多少个团队

既然管理多个 Scrum 团队这么困难，那我们干嘛还要找罪受呢？为啥不把所有人都放到一个团队里面去呢？

在我们曾有过的团队中，单个 Scrum 团队最多包括 11 个人。大家可以一起工作，但是效果不好。每天的 Scrum 会议基本上都会超过 15 分钟。每个人都不太清楚其他人在做什么，所以整个状态就有些混乱。Scrum master 很难保证每个人都在向同一个目标努力，也不太能找得到时间来解决发现的所有问题。

有人可能会建议说，把大团队分成两个团队。但这样做情况就一定会好转么？未必。

火龙果 · 整理
uml.org.cn

如果这个团队在实施 Scrum 方面很有经验，也习惯这种做法，而且能够以符合其内在逻辑的方式切分产品，把它分成两个独立的部分，保证各自的源代码不会重叠，那把团队分割就是一个好主意。不然我还是会坚持用一个团队，尽管大团队存在种种缺陷。

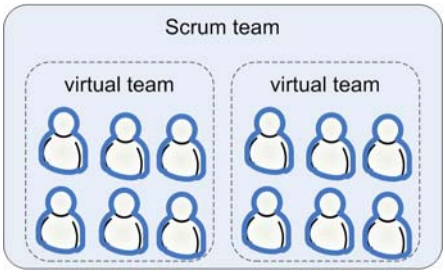
我的经验是，宁可团队数量少，人数多，也比弄上一大堆总在互相干扰的小团队强。要想拆分小团队，必须确保他们彼此之间不会产生互相干扰！

虚拟团队

在“大团队”和“多团队”之间权衡利弊之后，你做出了自己的决策，可怎么知道这种决策是对还是错呢？

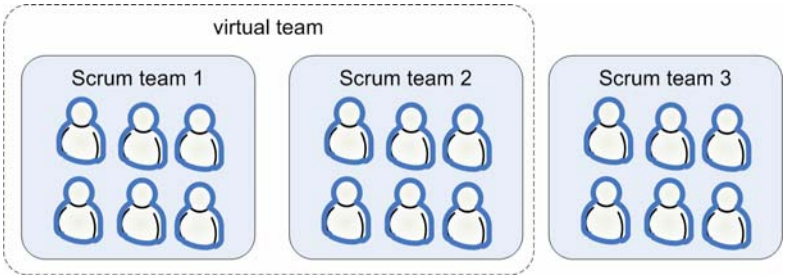
如果注意观察、仔细聆听，也许你会注意到 “虚拟团队”的存在。

例 1：你选择了使用“大团队”。不过观察一下 sprint 中的交流方式，你就能发现事实上这个大团队自动分成了两个子团队。



例 2：你选择了使用三个小团队的方式。不过观察一下 sprint 中的交流方式，你就会发现团队 1 和团队 2 一直在交流，而团队 3 比较孤立。

火龙果 · 整理
uml.org.cn



那么这到底意味着什么呢？是你的团队分割策略有问题吗？唔，如果类似虚拟团队一直这样保持下去的话，那就表示做错了；如果只是暂时的话，那就没问题。

让我们再看一下例 1。如果这两个虚拟的子团队一直变化（也就是大家在虚拟团队中换来换去），那把他们放到一个团队中就没有问题。如果二者的构成在整个 sprint 中保持不变，在下个 sprint 中可能就得考虑把他们分成两个真正的 Scrum 团队了。

现在再看看例 2。如果团队 1 和团队 2 在整个 sprint 中一直聊来聊去（把团队 3 扔在一边），在下个 sprint 中你大概就得把团队 1 和 2 合并到一块。如果在 sprint 的前半阶段，团队 1 和团队 2 一直交流，然后在后半阶段，团队 1 和团队 3 又相谈甚欢，那合并或者保持原样就都是可行的。你可以在 sprint 回顾会议上提出这个问题，让团队自己决定。

在 Scrum 中，团队分割确实很困难。不要想的太多，也别费太大劲儿做优化。先做实验，观察虚拟团队，然后确保在回顾会议上有足够的时间来讨论这种问题。迟早就会发现针对你所在环境的解决方案。需要重视的是，必须要让团队对所处环境感到舒适，而且不会常常彼此干扰。

最佳的团队尺寸

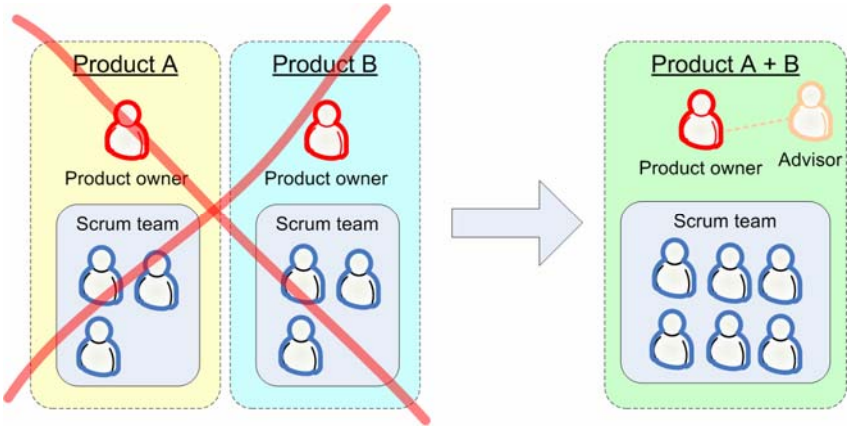
在我读过的大多数书中，5 到 9 个人被公认为是“最佳的”团队构成人数。

火龙果 · 整理
uml.org.cn

从到目前为止观察到的情况来看，我同意这种说法。不过我会建议说 3 到 8 个人。而且我相信，为了达到这种团队规模，花上一定代价还是值得的。

假设你有一个 10 人的 Scrum 团队。那么就考虑一下把最差的两个人踢出去吧。噢，我真的这么说过么？

假设你有两个不同的产品，每个产品都由一个 3 人团队负责，进度都很慢。**也许**可以把他们合并成有 6 个人的团队，同时负责这两个产品。然后让其中一个产品负责人离开这里（或者给他顾问之类的角色）。

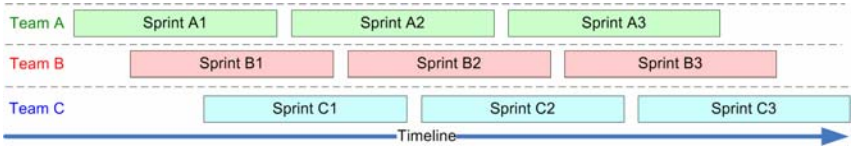


假设你的团队有 12 个人，因为代码库很烂，所以两个团队不可能独立在上面工作。那就应该认真投入时间、精力修复代码库（而不是引入新特性），直到可以分拆团队为止。这种投资很快就可以得到回报。

是否同步多个 Sprint？

假设有三个 Scrum 团队开发同一个产品。他们的 sprints 应该同步吗？在同样的时间启动和停止？或者应该交叉覆盖？
我们一开始是让这些 sprints 有交叉（考虑到各自的时间安排）。

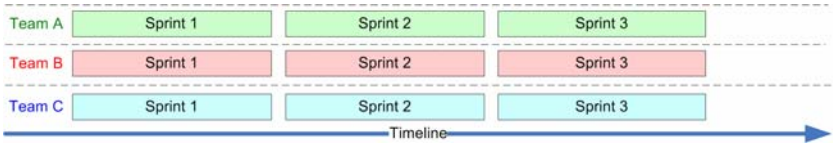
火龙果 · 整理
uml.org.cn



听上去挺不错。在任何一个给定的时间点上，都有一个正在进行的 sprint 接近结束，而新的 sprint 即将开始。产品负责人的工作负担会随着时间的推移逐步摊开。各个版本如溪水般汩汩流出。每周都有演示。老天保佑！

耶，我知道你想说什么，但大家**从前**确实觉得这个想法挺不错的！

我们一开始也是这么做的，直到有一天我有机会跟 Ken Schwaber（在我的 Scrum 认证期间）进行了交流。他指出这种做法**很有问题**，如果将各个 sprint 同步的话会好得多。我记不清他的确切理由，但经过几次讨论之后我就被他说服了。



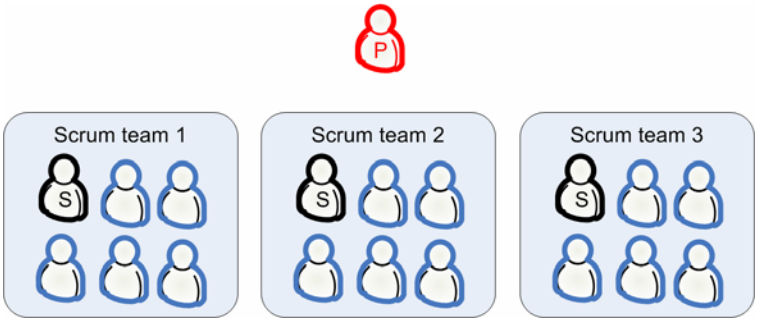
从那以后我们就采用了上图所示的解决方案，也从没觉得有什么不对劲儿的。我也没机会了解那种交叉的方案是否终会失败，但我觉得应该如此。同步进行的 sprint 有如下优点：

- 可以利用 sprint 之间的时间来重新组织团队！如果各个 sprint 重叠的话，要想重新组织团队，就必须打断至少一个团队的 sprint 进程。
- 所有团队都可以在一个 sprint 中向同一个目标努力，他们可以有更好的协作。
- 更小的管理压力，即更少的 sprint 计划会议、sprint 演示和发布。

火龙果 · 整理
uml.org.cn

为什么我们引入“团队领导”的角色

假设我们有三个团队开发同一个产品。



那个标记为“P”的红色家伙是产品负责人。标记为“S”的黑色家伙是Scrum Masters。其他的就是~~一直哼哼唧唧的~~……呃……值得尊敬的团队成员。

在这个群星荟萃的团队中，谁来决定哪些人属于哪个团队？产品负责人？三个 Scrum masters 集体决定？还是每个人都可以选择自己的团队？那如果每个人都想待在团队 1 里面怎么办（因为第一个 Scrum master 长得太**好看**了）？

如果后来发现最多只能有两个团队并行工作在这个代码库上，那我们就得把这三个 6 人团队变成 2 个 9 人团队。那当前这 3 个 Scrum masters 中，哪一个会失去头衔？

很多公司都有这种敏感问题。

有人可能会觉得让产品负责人来做人员分配是个好主意。但这不是产品负责人职责以内的事情，对吧？产品负责人是领域专家，他可以指导团队的前行方向，但不应该被牵扯到乱七八糟的扯淡细节中。尤其是如果他是“chicken”的话（如果你不了解 chicken 和 pig 的隐喻，可以 google 一下“chickens and pigs”）。

火龙果 · 整理
uml.org.cn

我们通过引入“团队领导”的角色来解决了这个问题。你也许把他叫做“Scrum 中的 Scrum master”，或者“老大”，也或者“首席 Scrum master”等。他不用领导某个团队，但是会负责跨团队的问题，例如谁担任哪个团队的 Scrum master，大家如何分组等等。

我们在给这个角色取名字的时候费了好大劲。我们找到了很多名字，“团队领导”已经算是最好的了。

这种方法效果很好，所以我也向你们推荐一下（怎么给这个角色命名就无所谓了）。

我们怎样在团队中分配人手

有多个团队开发同一个产品时，一般有两种分配人手的策略。

- 让一个指定的人来做分配，例如我前面提到的“团队领导”，或产品负责人，或职能经理（如果他的参与度比较高，就可以做出正确的决定）。
- 让团队自己决定。

我们这三种全都用过。三种？是的，策略 1，策略 2，还有二者的组合。

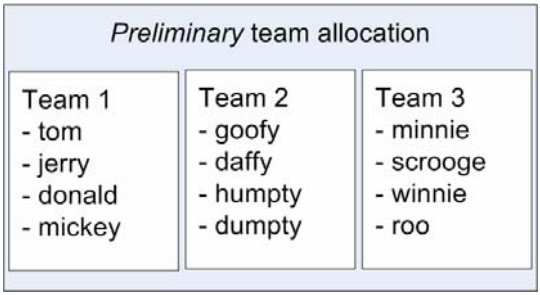
我们发现二者组合以后的效果最好。

在 sprint 计划会议之前，团队领导会跟产品负责人和所有的 Scrum masters 一起开团队分配会议。我们共同讨论上一个 sprint，决定是否需要进行重分配。也许会合并两个团队，或者调换某个人。我们就一些问题达成一致，并写到**团队分配提案**中，在 sprint 计划会议上进行讨论。

在 Sprint 计划会议上，我们首先遍历产品 backlog 中优先级最高的条目。然后团队领导说：

“各位，我们建议下一个 sprint 这样分配人手。”

火龙果 · 整理
uml.org.cn



“你们看，我们会从 4 个团队变成 3 个。每个团队中的人员名单已列出来了。你们可以凑到一块，自己商量一下要墙上的哪块地方。”

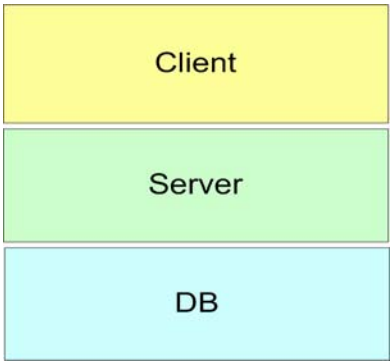
（团队领导耐心地等着大家在房间里转悠，直到他们分成 3 组，各自站在一块空墙下）。

“目前这个团队分配只是**初步计划**！就是为了节省点大家的时间。接下来开会的时候，你们还可以去另一个团队，或者把你们这个团队一分为二，或者跟另一个团队合二为一，怎么都行。做选择的时候动动脑子，考虑一下产品负责人定下来的优先级。”

我们发现这种方式效果最好。最开始使用一些集中式控制，然后再用分散式优化。

是否使用特定的团队？

假设你们的技术选型包括三种主要组件：

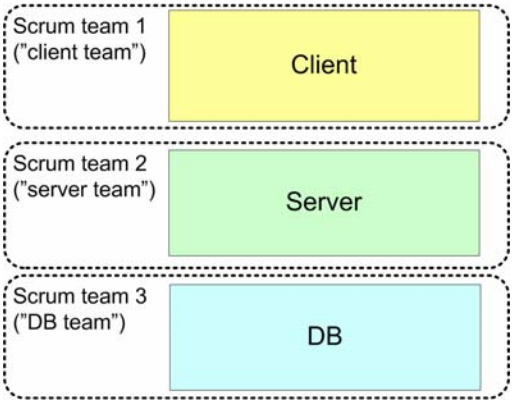


火龙果 · 整理
uml.org.cn

再比如说有参与开发这个产品的有 15 个人之多，所以你也不想把他们都放在一个 Scrum 团队里面。那该怎样创建团队呢？

方式 1：特定于组件的团队

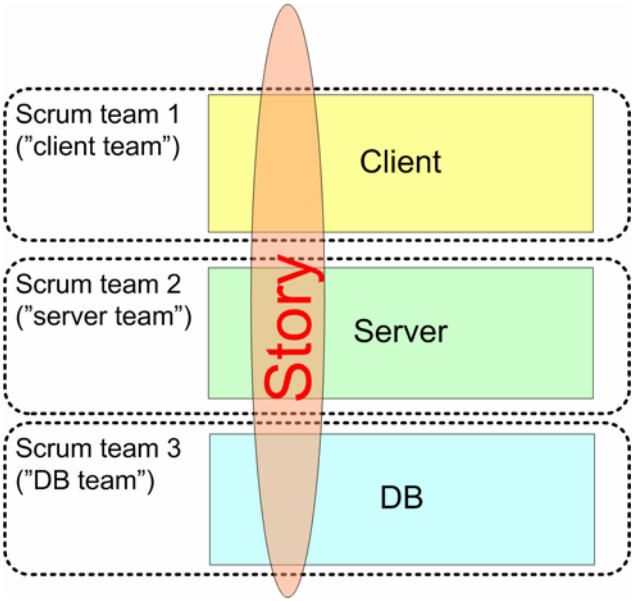
方式之一是创建针对特定组件展开工作的团队，例如“client 团队”、“server 团队”和“DB 团队”。



我们以这种方式开始。但效果不太好，要是大多数故事都涉及到多个组件就更糟了。

比如：如果有一个名为“留言板，可供用户在上面给彼此留言”的故事。这个特性需要更新客户端的用户界面，向服务器中添加逻辑，还要增加数据库中的表。

火龙果 · 整理
uml.org.cn

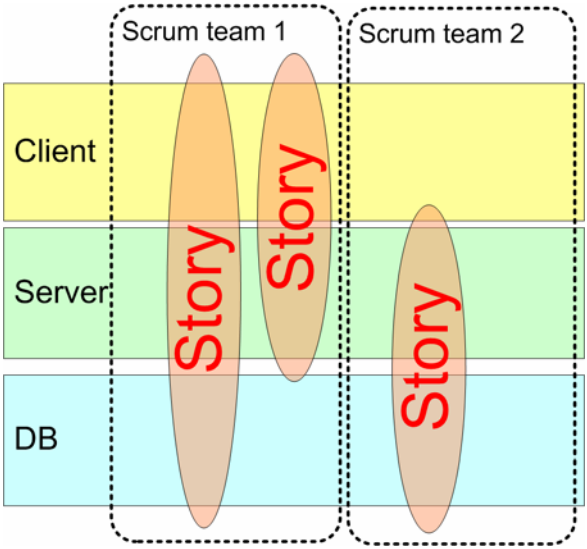


这就意味着这三个团队 – client 团队、server 团队和 DB 团队需要协作来完成这个故事。情况不妙啊。

方式 2：跨组件的团队

第二种方式是创建跨组件的团队，也就是说团队的职责不会被束缚在任何特定的组件上。

火龙果 · 整理
uml.org.cn



如果大多数故事都包括多个组件，那这种团队划分方式的效果就很好。每个团队都可以自己实现包括 client、server 和 DB 三部分的完整故事。他们可以互相独立工作，这就**很好**。

我们在实施 Scrum 的时候，所做的第一件事情就是打乱特定于组件的团队（方式 1），创建跨组件的团队（方式 2）。它减少了诸如“我们没法完成这个条目，因为我们在等 server 那帮家伙完成他们的工作”之类的情况发生。

不过，要是有很强烈的需求，我们也会临时创建针对特定组件展开工作的团队。

火龙果 · 整理
uml.org.cn

是否在 sprint 之间重新组织团队?

一般来讲，由于各自优先级最高的故事类型不同，不同的 sprint 之间会有很大差别；因此也会导致各个 sprint 理想的团队构成也有所不同。

实际上，几乎在每个 sprint 中我们都会发现自己在说："**这个** sprint 确实非同一般，原因在于……"一段时间以后，我们就放弃了"普通"sprint 的观念。世界上没有普通的 sprint，就像没有"普通"的家庭和"普通"的人一样。

在 sprint 中，组建一个只负责客户端的团队，团队中每个人都熟悉客户端代码，这也许是个好主意。到了下个 sprint，也许弄两个跨职能团队，把负责客户端代码的人拆分出去也是个好主意。

"团队凝聚力"是Scrum的核心要素之一，如果一个团队合作工作达多个sprint之久，他们就会变得**非常紧密**。他们会学会如何达成团队涌流（*group flow*）[请参见 http://en.wikipedia.org/wiki/Flow_(psychology)，译者注]，生产力会提升至难以置信的地步。不过要达到这个地步需要花上一定时间。如果不断变换团队组成，你就永远无法得到强悍的团队凝聚力。

所以，如果你确实想要重新组织团队，请先考虑一下后果。这是个长期变化还是短期变化？如果是短期变化，最好考虑跳过这一步。如果是长期变化，那就干吧。

这里有个例外：第一次在大型团队中开始实施 Scrum 的时候，你需要就团队拆分进行一些实验，最后才能找到令所有人全都满意的做法。要确保所有人都能够理解：在最开始几次时犯些错误是可以接受的，只要能够持续改进。

火龙果 · 整理
uml.org.cn

兼职团队成员

我很认同 Scrum 书中所说的话——在 Scrum 团队中含有兼职成员一般都不是什么好主意。

假设 Joe 是 Scrum 团队中的兼职成员。在让他进团队之前，你最好先认真考虑一下：这个团队确实需要 Joe 么？你确定 Joe 不能全职工作？他还要做什么其它事情呢？能不能找其他人接过 Joe 的其他工作，让 Joe 在那份工作中只起到被动的、支持性的作用？Joe 能不能从下一个 sprint 起在你的团队中全职工作，同时把他的其他工作转交给其他人？

有时就是没有其他办法。你没有 Joe 不行，因为他是这个楼里唯一的 DBA，但是其他团队也非常需要他，所以他永远不可能把所有的时间都分配给你的团队，而公司也不能雇用其他 DBA。好吧。这种情况下就可以让他兼职工作了（这恰恰是我们碰到的情况）。但你要确定每次都进行这种评估。

一般来讲，我宁愿要三个全职工作的成员，也不愿意要 8 个只能做兼职的。

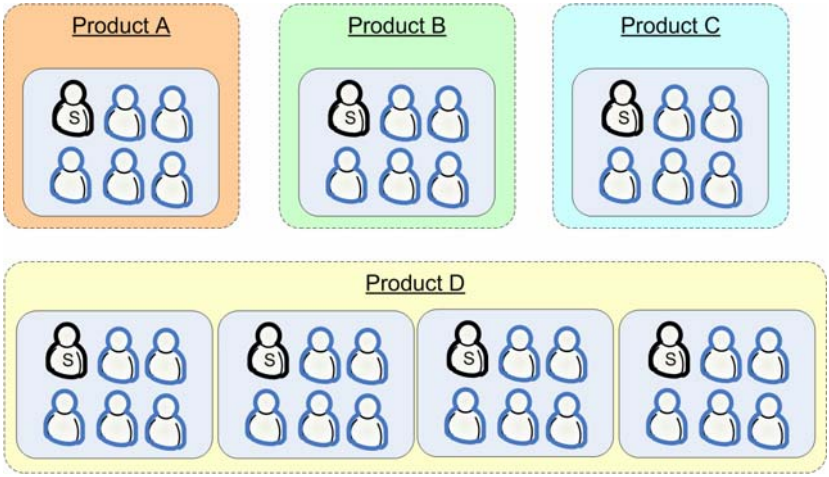
如果有一个人需要把他的时间分配给多个团队，就像上面提到的 DBA 一样，那最好让他有一个主要从属的团队。找出最需要他的团队，把它当作他的“主队”。如果没有其他人把他拖走，那他就得参加这个团队的每日 scrum 会议、sprint 计划会议、回顾等等。

我们怎样进行 Scrum-of-scrums

Scrum-of-scrums 实际上是一个常规会议，是为了让所有的 Scrum master 聚到一起交流。

我们曾经有过四个产品，其中三个都只有一个 Scrum 团队，而最后一个产品则共有 25 人，分成了好几个 Scrum 团队，如下图所示：

火龙果 · 整理
uml.org.cn



这意味着我们有两个层次的 Scrum-of-Scrums。一个是“产品层次”的 Scrum-of-Scrums，包括 Product D 中的所有团队，另外一个是“团体层次”的 Scrum-of-Scrums，包括所有的产品。

产品层次的 Scrum-of-Scrums

这个会议非常重要。我们一周开一次，有时候频率会更高。在会议上我们会讨论集成问题，团队平衡问题，为下个 sprint 计划会议做准备，等等。我们为此分配了 30 分钟时间，但常常超时。其实也可以每天进行 Scrum-of-Scrums，但我们一直没有时间尝试。

我们的 Scrum-of-Scrums 议程安排如下：

1) 每个人围着桌子坐好，描述一下上周各自的团队都完成了什么事情，这周计划完成什么事情，遇到了什么障碍。
2) 其他需要讨论的跨团队的问题，例如集成。

Scrum-of-Scrums 的议程对我而言无关紧要，关键在于你要有定期召开的 Scrum-of-Scrums 会议。

火龙果 · 整理
uml.org.cn

团体层次的 Scrum-of-Scrums

我们把这个会议称为“脉动”。我们试过多种形式，参与者也多种多样。后来就放弃了整个概念，换成了每周的全体（嗯，所有参与开发的人）会议。时长 15 分钟。

什么？15 分钟？全体参加？每一个产品所包括的全部团队中的所有人都会参加？这能行么？

是的，能行。只要你（或是其他主持会议的人）严格限定会议的时间不要过长。

会议的形式为：
1) 开发主管介绍最新情况。例如即将发生的事件信息。
2) 大循环。每个产品组都有一个人汇报他们上周完成的工作，这周计划完成的工作，及碰到的问题。其他人也会作报告（配置管理领导，QA 领导等）
3) 其他人都可以自由补充任何信息，或者提问问题。

这是个发布概要信息的论坛，而不是提供讨论或者反映问题的场所。只要保证这一点，15 分钟常常就足够了。有时我们也会超时，但极少会占用 30 分钟以上的时间。如果出现了热烈的讨论，我就会打断它，请感兴趣的人在会后留下继续讨论。

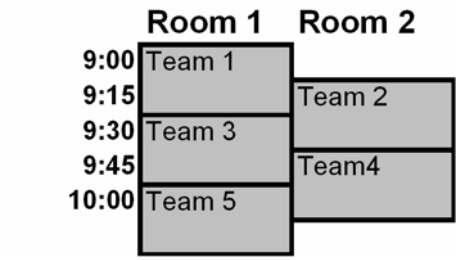
为什么我们要进行全体的脉动会议呢？因为我们发现团体层次上的 Scrum of Scrums 主要以报告形式进行，很少出现真正的讨论。另外，在这个圈子以外，有许多人都对这种信息非常感兴趣。基本上大家都想知道其他团队在做些什么。所以我们想，既然已经打算聚到一起，花时间来告诉彼此每个团队都在干什么，那为什么不让所有人都参加呢。

火龙果 · 整理
uml.org.cn

交错的每日例会

如果有太多的 Scrum 团队参与单个产品的开发，而且他们都在同一时刻进行每日例会，那你就遇到问题了。产品负责人（以及像我一样爱管闲事的家伙）因此每天只能参加一个团队的每日例会。

所以我们要求团队避免在同一时刻进行每日例会。



上面的例子中，我们是这样安排的：每日例会不在团队房间中进行，而是安排在不同的房间。每个会议大约 15 分钟，但是每个团队在房间中都可以使用 30 分钟的时间，以备他们会稍稍超出一点儿时间。

这种做法**超级有效**，原因有二：

1. 像产品负责人和我这样的人可以在一个早上参加**所有的**例会。想清楚了解到当前的 sprint 进展状况，有什么严重的风险，这是最好的方式。
2. 团队成员可以参加其他团队的例会。这种情况不常发生，不过有时两个团队会在相似的环境下工作，所以会有几个人参加彼此的例会来保持同步。

它的缺点是减少了团队的自由度——他们无法选择他们自己喜欢的时间开例会。不过这一直没成为我们的问题。

火龙果 · 整理
uml.org.cn

救火团队

曾经有那么一次，在一个大型产品的开发过程中，我们实施不了 Scrum，因为团队成员花了太多时间来救火——拼命忙着修复早期版本中的 bug。这是个恶性循环，影响很坏，他们花了太多时间救火，最后根本没有时间进行前瞻性的工作来防火（改进设计、自动化测试、创建监控工具与警报工具等）。

我们创建了一个专门的救火团队，一个专门的 Scrum 团队，从而解决了这个问题。

Scrum 团队的工作是（带着产品负责人的祝愿）稳定系统，有效防火。

救火团队（实际上我们管他们叫“支持团队”）有两项工作。
1) 救火。
2) 保护 Scrum 团队远离各种干扰，包括挡开那些不知从何而来的、增加临时特性的要求。

救火团队被安排在离门最近的地方，Scrum 团队坐在房间的最里面。所以救火团队可以真正地**从物理上保护** Scrum 团队，使他们不会受到急切的销售人员或者怒气冲冲的客户的干扰。

两个团队中都有高级工程师，这样一个团队就不会过于依赖另一个团队的核心人员。

这实际上也是对解决 Scrum 自行启动问题的一种尝试。如果团队的工作计划总是只能安排一天之内的工作，那我们怎么开始做 Scrum 呢？就像上面所讲述的那样，我们的策略是分割团队。

这种方式效果很好。因为 Scrum 团队有了空间努力工作，所以他们最后能够稳定系统。同时救火队员也完全放弃了预先计划的想法，他们完全是针对外部反应展开工作，只管修复即将出现的下一个问题。

火龙果 · 整理
uml.org.cn

当然，Scrum 团队也不是**完全**远离干扰。救火团队常常需要 Scrum 团队中核心人员的帮助，在最糟糕的状况下，甚至会需要整个团队。

但无论如何，经过几个月以后，这个系统达到了足够稳定的状态，然后我们解散了救火团队，另外创建了一个新的 Scrum 团队。救火队员们很高兴把已经磨损的头盔放到一边，加入到 Scrum 团队中。

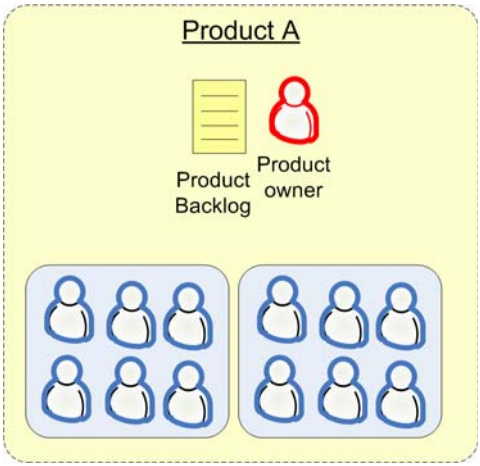
是否拆分产品 backlog？

假设你有一个产品和两个 Scrum 团队，那应该有几个产品 backlog 呢？多少个产品负责人？我们曾经为此评估过三个模型。选择不同，sprint 计划会议的形式就会有很大差异。

策略 1：一个产品负责人，一个 backlog

这就是“只能有一个”的模型，也是我们最推崇的模型。

这种模型的优点是：你可以让团队根据产品负责人当前的优先级来自行管理。产品负责人关注**他所需要的东西**，团队决定怎么分割工作。

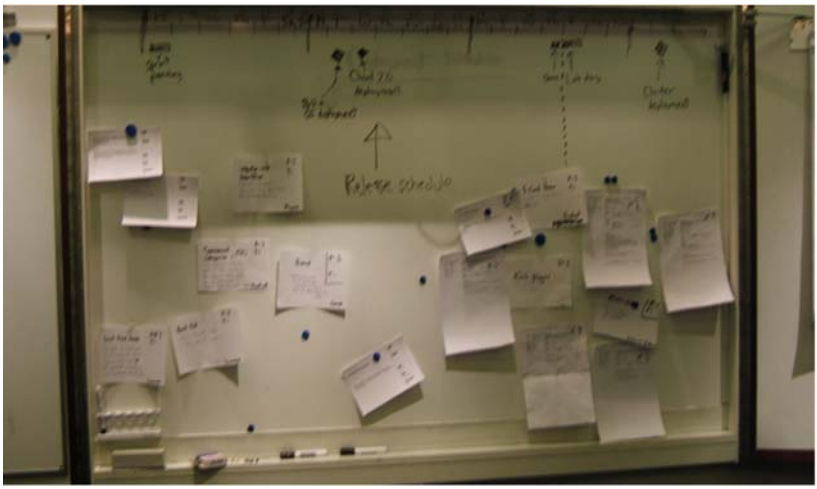


火龙果 · 整理
uml.org.cn

说得更具体一些，我们来看看这个团队 sprint 计划会议的举行方式：

Sprint 计划会议在一个外部的会议中心举行。

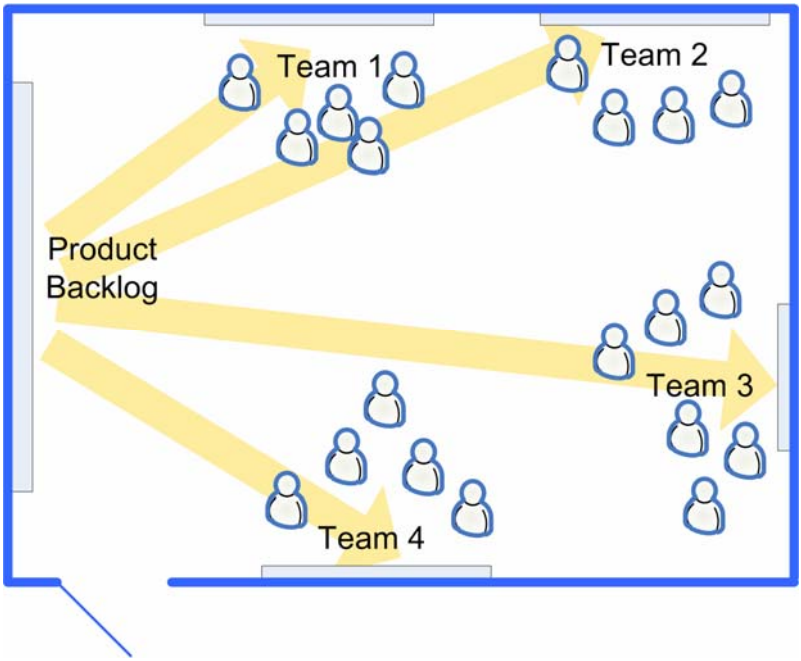
在会议开始之前，产品负责人指定一面墙壁用做“产品 backlog 墙”，把故事贴在上面（以索引卡的形式），按相对优先级的顺序排序。他不断往上面贴故事，直到贴满为止。通常他贴上去的东西都要比一个 sprint 中所能完成的条目多。



每个 Scrum 团队各自选择墙上的一块空白区域，贴上自己团队的名字。那就是他们的“团队墙”。然后他们从最高优先级的故事开始，从产品 backlog 墙上把故事逐一挪到他们自己的团队墙上。

这个过程可以用下面的图片来描述，图中的黄色箭头表示故事卡从产品 backlog 墙移动到团队墙的过程。

火龙果 · 整理
uml.org.cn



在会议进行中，产品负责人与团队会针对索引卡进行讨论、把它们在团队之间移动、上下挪动以调整优先级、把它们拆分成更小的条目，等等。过上大概一小时左右，每个团队就会在自己的团队墙上形成一个 sprint backlog 的初步候选版本。然后团队便会独立工作，进行时间估算，把故事拆分成任务。

火龙果 · 整理
uml.org.cn

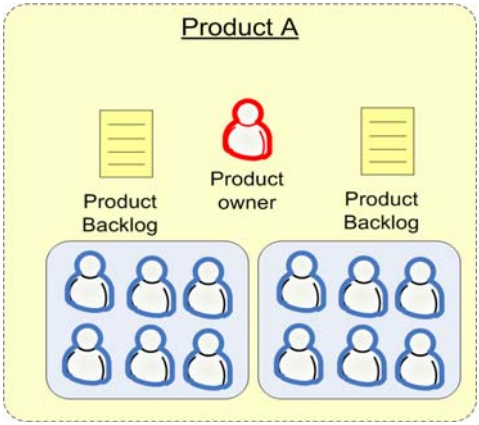
整个过程显得特别嘈杂混乱，令人筋疲力尽，但同样也效果很好，很有趣，也是个社会交往的过程。到结束时，所有团队通常都会得到足够的信息来启动他们的 sprint。

策略 2：一个产品负责人，多个 backlog

在这种策略中，产品负责人会维护**多个**产品 backlog，每个团队对应一个。我们没有真正试过这种方式，不过差点儿就这么做了。这是我们的后备方案，以防第一种策略失败。

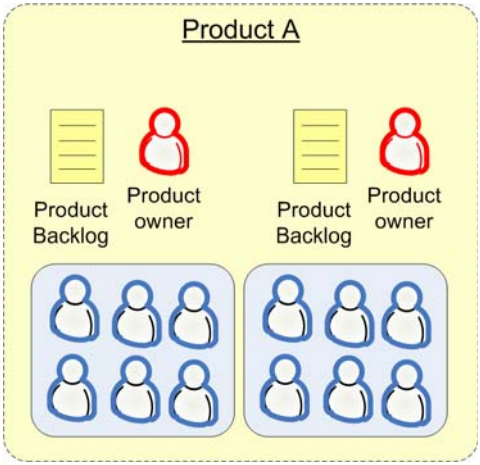
它的劣势在于，产品负责人要把故事分配给团队，而这项工作交给团队自己处理会更好。

火龙果 · 整理
uml.org.cn



策略 3：多个产品负责人，每人一个产品 backlog

它跟第二个策略有点像，每个团队都有一个产品 backlog，但每个团队也都有一个产品负责人！



我们没有用过这种方式，也许永远也不会用。

如果两个产品 backlog 都对应同一个代码库，那两个产品负责人可能会发生严重的利害冲突。

火龙果 · 整理
uml.org.cn

如果两个产品 backlog 所对应代码库不同，那这样做，就跟把整个产品分成不同的子产品然后独立运作毫无二致。也就表示着我们回到了每个团队一个产品的情况，这样处理起来既愉快又轻松。

代码分支

有多个团队在同一个代码库基础上工作，我们就势必会碰到 SCM（软件配置管理）系统中的代码分支问题。现在已经有很多关于处理多人协同工作问题的书和论文了；所以我这里也就不再谈什么细节。我也没有什么新东西或者革命性的观点，下面会总结一下我们团队到目前为止学到的最重要的一些经验。

- 主线（或者主干）的状态要严格保持一致：最起码所有的东西都要能够进行编译，所有的单元测试都可以通过。**每时每刻**都能创建一个可以工作的发布版本。如果可以做到持续构建系统在每晚进行构建，并把结果自动部署到测试环境中就更好了。
- 给每个版本打上标记（tag）。无论什么时候，只要是为验收测试进行发布，或是发布到产品环境，在主线中就应该进行版本标记，用来精确标识所发布的内容。这便意味着在将来的任一时刻，你都可以回退到某个历史版本中，创建一个维护分支。
- 只在必需的时候创建分支。这里有一条很好的规则：如果你无法在不违反现有代码基线策略的情况下使用该代码基线，那么**只有**在这种情况下，才能创建新的代码基线。如果摸不准是什么情况，那就不要创建分支。为什么？因为每个活动分支都会增加复杂性，提高管理成本。
- 将分支主要用于分离**不同的生命周期**。无论你是否决定让每个团队在他们自己的代码基线上进行编码，如果你打算在同一个代码基线上将短期的修复版与长时间的变化进行合并，到时候就会发现：要发布这个短期的修复版绝非易事！
- 经常同步。如果你在分支上工作，那么只要有了一些代码可以构建，就应该与主线进行同步。在每天的编码工作开

火龙果 · 整理
uml.org.cn

始之前，都把代码从主线同步到分支上，这样你的分支就可以与其他团队所做出的变化保持更新。如果会产生让你觉得生不如死的合并情况，那也只能接受这种现实，因为等下去的结果只会更糟。

多团队回顾

如果有多个团队开发同一个产品，我们怎样做 sprint 回顾呢？

在 sprint 演示结束以后，大家鼓掌、相拥，然后每个团队立刻回到自己的房间，或者办公室之外的某个舒适场所。他们各自的回顾方式与我在“我们怎样做 sprint 回顾”中描述的情况并没什么不同。

在 sprint 计划会议上（因为我们在同一个产品中使用的是同步的 sprint，所以所有团队均会参加），第一件事情就是让每个团队中找出一个发言人，站起来总结他们回顾中得出的关键点。每个团队都有 5 分钟的时间。然后我们会进行大约 10 到 20 分钟的开放讨论。之后稍作休整，开始真正的 sprint 计划。

我们没有试过其它方式，这样已经足够了。不过最大的缺点就是在回顾之后，计划会议之前没有休整时间（参见“sprints之间的休整时刻”）。

对于单个团队的产品，我们就不会在 sprint 计划会议上对回顾进行总结了。因为这没有必要，每个人都参与了真正的回顾会议。

火龙果 · 整理
uml.org.cn

16

我们怎样管理地理位置上分布的团队

如果团队成员处于不同地理位置该怎么办？Scrum 和 XP 的大部分“魔力”要想发挥作用，团队的成员们最好身处同地紧密协作、可以结对编程，而且能做到每日面对面交流。

我们有一些分散的团队，也有些团队成员时时在家工作。

我们的策略很简单：就是想尽办法来把物理位置上分散的团队成员之间的沟通带宽增至最大。我不只是说每秒传递多少兆字节（当然这也很重要），还包括含义更广的沟通带宽：

- 能够一起结对编程。
- 能够在每日例会上面对面交流。
- 在任何时候都能够面对面讨论。
- 可以真正地碰面与交往。
- 整个团队可以主动举行会议。
- 团队对 sprint backlog、sprint 燃尽图、产品 backlog 和其他信息传递设施有相同的理解。

我们还采取过其他一些措施（或者是正在试着实施，到现在还没有全都用到过）：

- 每一台工作站前面都配备网络摄像头和耳麦。
- 可以远程通话的会议室，带有网络摄像头、会议用麦克风、随时可用的计算机和桌面共享软件等等。

火龙果 · 整理
uml.org.cn

- “远程窗口”。每个地方都有大屏幕，显示其他地点的固定画面。就像两个公寓之间的虚拟窗口一样。你可以看到谁坐在座位前，谁在跟谁说话。这可以增强“我们是在一起工作”的感觉。
- 交换程序。来自每一个地方的人按照某个规律交叉访问。

通过类似技术以及更多手段，我们可以慢慢掌握到，如何在地理分布的团队之间开展 sprint 计划会议、演示、回顾和每日 scrum 会议等等。

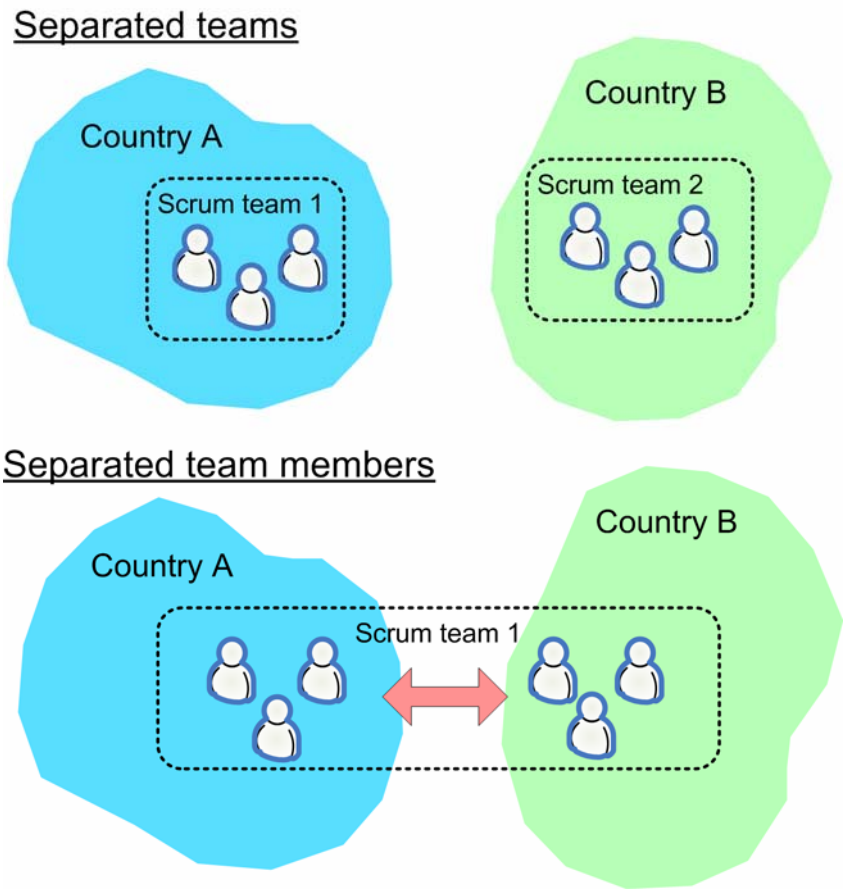
和其他规律一样，这也是通过不断的实验总结出来的。观察 =>调整=>观察=>调整=>观察=>调整=>观察 =>调整=>检查 =>调整

离岸

我们也有一些离岸团队，并且尝试过如何用 Scrum 来提高协作效率。

离岸的方式主要分为两种：分散的团队和分散的团队成员。

火龙果 · 整理
uml.org.cn



第一种方式是被迫下的选择。不过我们还是先以第二种方式开始离岸开发的。原因如下：

1. 我们希望团队成员可以对彼此有很好的了解。
2. 我们希望在两地之间能够有良好的沟通基础，也想让团队有强烈的愿望来把基础打好。
3. 在刚开始的时候，离岸团队比较小，没法自己组成一个有效的 scrum 团队。

火龙果 · 整理
uml.org.cn

4. 在独立离岸团队可以正常运作之前，我们要有一段紧张忙碌的信息共享时期。

从长期来看，我们也许会顺利过渡到“分散的团队”这种方式上去。

在家工作的团队成员

在家工作有时候会效果很好。有时，你在办公室里一星期也干不完的工作，在家里一天就搞定了。只要你没有孩子的话:o)

不过，团队应该处在相同的物理位置是 Scrum 的基本原则之一。那我们是怎么做的呢？

通常我们让团队自己决定在家工作的时间和频率。有些团队成员因为家和办公室的距离太远，所以常常在家工作。不过我们还是鼓励团队在“大多数”时间尽量聚在一起。

在家工作时，他们会通过 Skype 语音通话来参加每日 scrum 会议。他们整天都保持在线，可以进行实时通信。虽然比起在同一个房间里还是有差距，但这也不错了。

我们曾经试过把星期三作为**聚焦日**。这表示“如果你想在家工作，这没问题，不过只能在星期三。而且要得到团队许可。”这种做法很有效。通常大多数人都会在星期三待在家里，完成大量工作，同时还能协作的很好。因为这只有一天，所以团队成员不会脱离彼此同步的状态太久。不过由于某些原因，这种做法从来都没有在其他团队中流行起来。

总的来说，团队成员在家工作，对我们基本上不是问题。

火龙果 · 整理
uml.org.cn

17

Scrum master 检查列表

在最后这章，我会展示一下我们的 scrum master“检查列表”。它列出了我们的 scrum master 日常执行的常用管理事务。这些都很容易被人们忘记。有些很显而易见的事情我们就略过不提了，如“消除团队的障碍”。

sprint 开始阶段

- Sprint 计划会议之后，创建 Sprint 信息页面。
  - 在 wiki 上创建从 dashboard 指向所创建页面的链接。
  - 把页面打印出来，贴在通过你们团队工作区域之外的墙上，让经过的人都可以看到。
- 给每个人发邮件，声明新的 sprint 已经启动。邮件中要包括 sprint 目标和指向 sprint 信息页面的链接。
- 更新 sprint 数据文档。加入估算生产率、团队大小和 sprint 长度等等。

每一天

- 确保每日 Scrum 会议可以按时开始和结束。
- 为了保证 sprint 可以如期完成，需要适当地增删故事。

火龙果 · 整理
uml.org.cn

  - 确保产品负责人了解这些变化。
- 确保团队可以及时得知 Sprint backlog 和燃尽图的最新状况。
- 确保存在的问题和障碍都能被解决，并报告给产品负责人以及（或者）开发主管。

在 sprint 结束时

- 进行开放式的 Sprint 演示。
- 在演示开始前一两天，就要通知到每个人。
- 与整个团队以及产品负责人一起开 Sprint 回顾会议。开发主管也应该受邀参加，他可以把你们的经验教训大范围传播开来。
- 更新 sprint 数据文档。加入实际生产率和回顾会议中总结出的关键点。

火龙果 · 整理
uml.org.cn

18

额外的话

喔！真没想过这本书会写到这么长。

无论你是初涉 Scrum，还是已饱经风霜，希望它都能带给你一些有用的想法。

因为 Scrum 必须针对每一种不同的环境来进行具体实施，所以很难站在通用的角度上讨论何谓最佳实践。不过无论如何，我都希望能够听到你的反馈。告诉我你的做法和我有什么区别。告诉我如何改进！

你可以通过 **henrik.kniberg@crisp.se** 来联系我。我同时也会常常关注 **scrumdevelopment@yahoogroups.com**。

如果你喜欢这本书，也许会对我的博客感兴趣。我也会在上面写一些有关 Java 和敏捷开发的话题：
http://blog.crisp.se/henrikkniberg/

哦，最后请不要忘记……

这只是一份工作而已，不是么？

火龙果 · 整理
uml.org.cn

推荐阅读

我的众多灵感与思想都来自于下面这些书。强烈推荐！

火龙果 · 整理
uml.org.cn

有关作者

Henrik Kniberg（henrik.kniberg@crisp.se）是一名咨询师，在斯德哥尔摩的Crisp公司（www.crisp.se）工作。他的专长是Java和敏捷软件开发。

自从第一本有关 XP 的书籍和敏捷宣言问世以来，Henrik 就开始拥抱敏捷原则，并尝试在不同的组织中进行有效应用。在 1998 年至 2003 年间，他作为 Goyada 的合作创始人和 CTO，构建并管理一个技术平台和 30 人的开发团队，充分试验了测试驱动开发及其它敏捷实践。

在 2005 年末，Henrik 签约了瑞典的一家游戏行业公司，作为该公司的开发部门主管。当时该公司形势危如累卵，组织管理及技术方面的问题极其严峻。通过使用 Scrum 和 XP，Henrik 将敏捷和精益原则贯彻到了公司的各个方面，帮助公司走出了困境。

2006 年十一月的一个星期五，Henrik 因为发烧生病，在家卧床。他决定记录下在过去的几年中所学到的知识。不过一经启动，他就再难搁笔，经过三天的疯狂之后，这份最原始的记录已经扩张成了一份 80 页的长文，名为“硝烟中的 Scrum 和 XP”，最后则形成了这本小书。

Henrik 走了一条全面发展的道路，他在各种角色之间怡然自乐：经理、开发人员、Scrum master、教师与教练。他一直致力于帮助公司构建优秀软件与优秀团队，充当各种必需的角色。

Henrik 在东京长大，目前与他的妻子 Sophia 和两个孩子生活在斯德哥尔摩。他在空闲时间还是一个活跃的音乐家，跟本地乐队一起创作乐曲，玩贝司和键盘。

这个网站上有他的更多信息：
http://www.crisp.se/henrik.kniberg