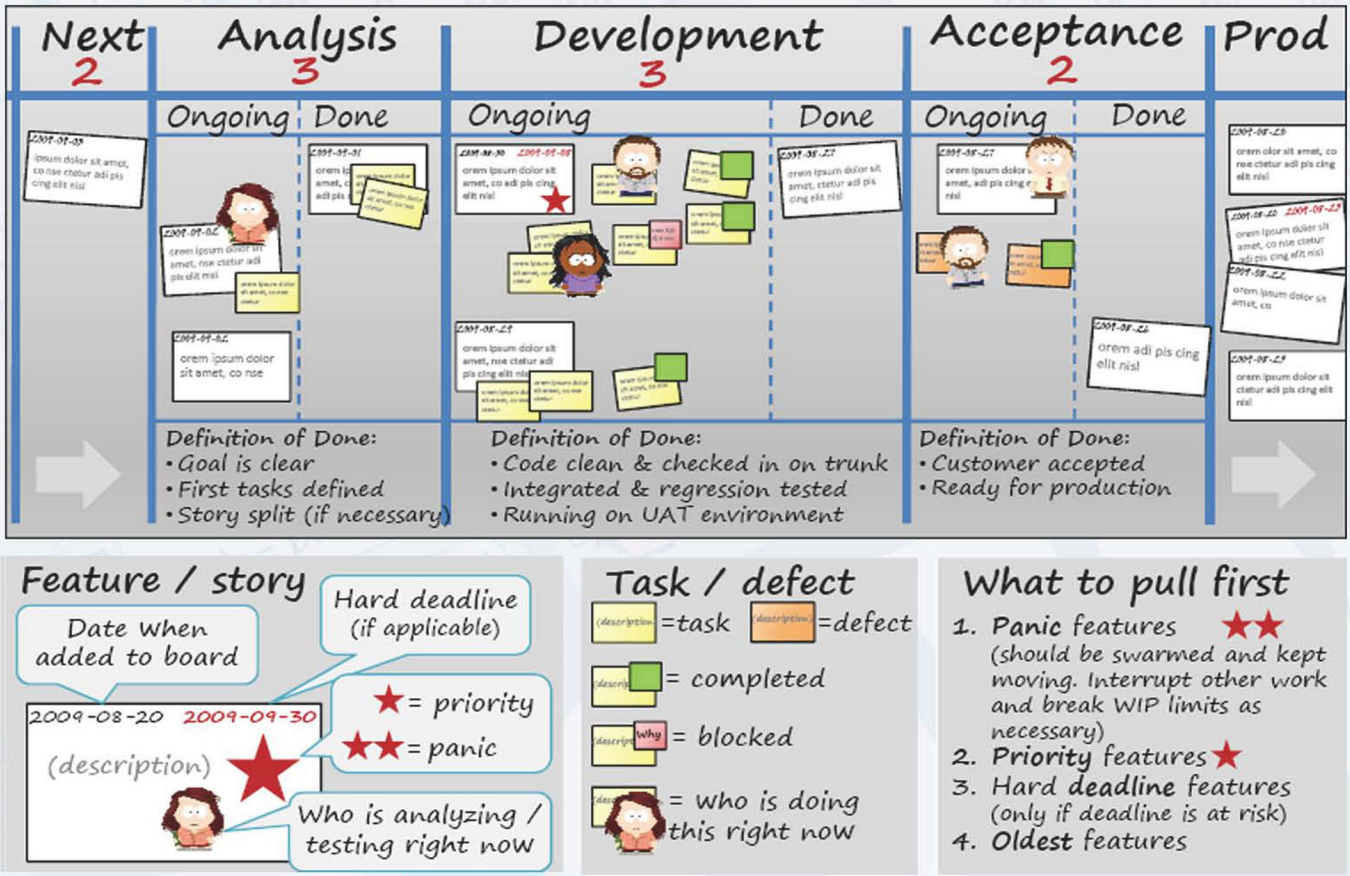


看板和Scrum

相得益彰

Henrik Kniberg、Mattias Skarin 著

Mary Poppendieck、 David Andersan 序

李剑 译

ENTERPRISE SOFTWARE DEVELOPMENT SERIES

InfoQueue

免费在线版本

（非印刷免费在线版）

了解本书更多信息请登录本书的官方网站

InfoQ 中文站出品

本书由 InfoQ 中文站免费发放，如果您从其他渠道获取本书，请注册 InfoQ 中文站以支持作者和出版商，并免费下载更多 InfoQ 企业软件开发系列图书。

本迷你书主页为

http://www.infoq.com/cn/minibooks/kanban-scrum-minibook-cn

看板和 Scrum——相得益彰

Henrik Kniberg、Mattias Skarin 著

Mary Poppendieck、David Anderson 序

李剑 译

郭晓刚 审校

C4Media, Publisher of InfoQ.com.

This book is part of the InfoQ Enterprise Software Development series of books.

For information or ordering of this or other InfoQ books, please contact books@c4media.com.

Managing Editor: Diana Plesa

Cover art: Bistrian Iosip

Composition: Accurance

Library of Congress Cataloguing-in-Publication Data:

ISBN: 978-0-557-13832-6

Printed in the United States of America

目录

InfoQ
Enterprise Software Development Community

时刻关注企业软件开发领域的变化与创新

我们的价值观：
创造可信赖的内容

我们的愿景：
帮助传播企业软件开发相关的知识和创新

我们的读者：
中高端技术人员，如架构师、项目经理、高级工程师等

我们的社区：
Java、.NET、Ruby、SOA、敏捷、架构和运维

我们的特质：
个性化RSS定制、国际化内容同步更新

我们的团队：
超过60位领域专家担当社区编辑

序—— Mary Poppendieck

Henrik Kniberg 的身上有一种稀有的特质，他可以从复杂的问题中剥离掉无关的因素，提取出最核心的思想，以最清澈最透明的方式讲述出来，让人觉得这些知识竟是前所未有地简单易懂。Henrik 在这本书里完成了一项壮举。他解释了 Scrum 和看板的区别，让读者明白它们不过是工具而已，我们所要做的是打造一个完备的工具箱，彻底弄懂每一项工具的长处和局限，明白什么情况下用什么工具。

在这本书中，你会了解到看板所要解决的问题域、它的强项和弱点、使用时机。你也会学到很精彩的一课——Scrum，或是你手头任何一款工具的改进时机和方案。Henrik 在书中讲得很明白，你一开始选用什么工具并不重要，重要的是你要不断改进它的用法，不断扩充你的工具箱。

Mattias Skarin 写了书的第二部分，他讲述了 Scrum 和看板在真实场景中的应用，把这本书变得更加精彩纷呈。这两个工具或是单兵突击，或是联合作战，推动着软件开发过程改进的步伐。书中讲到，世界上没有“最好”的方式；每个人都需要在自己的场景中思考，找到继续前行的方向。

Mary Poppendieck

序—— David Anderson

看板的本质是一个很朴素的思想：在制品（work-in-progress，WIP）必须被限制。只有当前的某项工作被交付，或是有了来自于下游的拉动，新的工作才能开始。看板（或信号卡）的含义是，因为当前的工作没到限额，有新任务可以拉进来，于是发出了一个肉眼可见的信号。这件事情听上去并不是革命性的变化，也似乎不会对团队和组织的绩效、文化、能力、成熟度产生多么深刻的影响。但它却奇迹般地做到了！！看板看似虽小，但却能影响业务中的一切。

我们意识到，看板是一种改变管理方针的途径。它不是软件开发、项目管理的生命周期或者流程。它是给现有的软件开发生命周期或者项目管理方法中引入变化的途径。实施看板的原则是把当前的工作作为起点，通过价值流分析来理解当前的流程，然后为每个环节中的在制品上限达成共识。让看板信号拉动着工作流动起来。

看板在敏捷软件开发的团队中成效显著，但也吸引了采用传统开发方式的团队的目光。在对组织文化进行精益改造，推动持续改善的过程中，看板往往会被先行引入。

因为 WIP 在看板中是受限制的，所以不管是什么任务，因为什么原因被阻塞，都会对整个系统造成严重影响。如果被阻塞的任务到了一定数量，整个流程就没法运作了。无论是团队还是组织都要聚焦于此，解决问题，让任务重新流动起来。

看板使用了可视化管理的方式，跟踪任务在整个价值流中流经的不同阶段。人们通常会用带贴纸的白板，或是电子卡片墙。我觉得最好的方式是两个都用。它带来的透明化也推动着文化上的改变。敏捷方法在某些方面已经有了很好的透明管理，例如在制品、已完成的工作、度量（如生产率——在一个迭代中完成的工作数量）；但看板做的要更加深入一些，它让流程中的所有环节以及工作的流动状态都完全呈现在我们眼前。它让人们看到了瓶颈、队列、变化、浪费。种种这些都会影响我们交付的有价值的成品数量，影响循环周期，从而影响组织效益。团队成员和外部相关干系人都可以通过看板看到自己的行为（和不作为）带来的影响。一些早期的案例分析表明，看板改变了人们的行为方式，促进了工作中更为紧密的协作。当人们看到瓶颈、浪费、变化所造成的影响之后，就会开始讨论如何做出改善，团队也会很快的把改善方案落实到过程中去。

由此可见，看板所提倡的是渐进式演化，逐渐向敏捷和精益的价值观靠拢。它并不要求像秋风扫落叶一般，把过去的工作方式一扫而光；它是随风潜入夜，润物细无声。它的改变是所有人彻底理解并达成共识的。

看板具有拉动系统的本质，提倡推迟承诺——不管是对新任务排序，还是交付当前手头的工作。一般来讲，团队会跟上游的相关干系人定期开会，排定接下来要做的工作。这种会议因为时间较短，所以会经常开。干系人在会上要回答一个很简单的问题，例如“到现在我们又有了两个空出来的位置。我们的循环周期是 6 周，你最希望在 6 周之后交付哪两项任务？”这个问题有双重含义。首先，一个简单的问题可以得到快速而明确的答复，缩短会议时间；其次，它还意味着我们要到了最后责任时刻才开始承诺接下来做什么事情。这可以更好地管理用户期望，缩短从承诺到交付的循环周期，而由于优先级变化的可能性减少，返工的几率也就少了。

我要说的最后一点是，制定 WIP 的限额可以让循环周期更容易预测，让交付更加可靠。出现阻碍或者 bug 之后的“停线”机制，可以确保质量水准，让返工情况急剧降低。

虽然书中的通透解释约略映照出一些成长印记，但我们已经没办法还原出看板完整的发展路径。看板不是在一个短短的午后时光里，靠着天赐般的灵感喷涌孕育出来的。它用了多年时间才渐渐成型。那些在文化、能力、成熟度、组织等各方面带来奇迹般改变的哲学和社会学因素都是我们所未曾想象过的，但它们切切实实地发生在了眼前。看板引发的很多结果都是反直觉的。看上去机械化的操作——WIP 上限和拉动式生产——却会对人与人之间交互、协作的方式产生长久深远的影响。不管是我，还是其他人，在刚开始使用看板的时候，都没有预想到这一点。

我曾经努力寻找一种阻力最小的改变手段，才有了后来的看板方法。那是 2003 年的事情。一开始我追求的是机械效益。后来在实施精益技术的时候发现，既然管理 WIP 是有意义的，那么限制它就更有意义了，这还能解放一部分管理的精力。于是 2004 年起，我就决定从几个首要原则开始推行拉动式系统。微软的一名经理找到我，问我能不能给他的团队引入一些改进，他们当时正在做内部 IT 系统的维护升级。我最开始用的是基于约束理论的拉动生产方案，Drum-Buffer-Rope。效果非常好：循环周期缩短了 97%，产出高了三倍，可预测性达到了 98%之多。

2005 年，Donald Reinertsen 劝我把看板全面落地。2006 年我得到了这样一个机会，当时我掌管着西雅图 Corbis 公司的软件工程部。2007 年，我开始展示实施看板的成果。最开始我是在 07 年芝加哥的精益新产品开发峰会上做了演讲。同年 8 月，又在 Agile 2007 大会的开放空间会议（open space）上进行了分享。当时有 25 人参加，三个来自于 Yahoo!。他们分别是 Aaron Sanders、Karl Scotland、Joe Arnold。他们各自回到加利福尼亚、印度和英国以后，就开始在挣扎着实施 Scrum 的团队中推行了看

板。他们还创建了一个 Yahoo!讨论组，在我写下这篇文章的时候，这个讨论组已经有了 800 个会员。看板正在被广泛传播，先行者开始了经验分享。

现在是 2009 年，看板的实施越来越普遍，我们看到了越来越多的一线报告。过去的五年里，我们从看板上面学到了很多东西，也会持续学习下去。我现在的工作重心就是实施看板、记录看板、讲述看板、思考看板，一切都是为了更好地理解看板，把它更好地传播给其他人。我渐渐地不再讨论看板和其他敏捷方法的区别，只在 08 年的时候花了些时间解释为什么看板可以跟敏捷兼容。

像“看板跟 Scrum 比起来怎么样？”这种问题，还是留给更有经验的人来回答吧。我很高兴 Henrik Kniberg 和 Mattias Skarin 成为了这方面的领军人物。作为知识工作者的你，需要信息辅助决策，指明道路。Henrik 和 Mattias 以我所不能及的方式满足了你的需求。我尤其欣赏 Henrik 幽默的文风，和他尊重事实、不固执己见的做事方式。他笔下的卡通像和图片要远远胜过长篇累牍的文字。Mattias 的案例分析也很重要，它用事实证明了看板不仅仅是理论而已，也许对你的组织会起作用。

我希望你会喜欢这本书。它会让你更加深刻地理解敏捷，尤其是看板和 Scrum。如果你想了解更多看板的知识，请访问我们的社区网站：The Limited WIP Society，http://www.limitedwipsociety.org/

David J. Anderson

于美国华盛顿州史魁恩镇

2009 年 7 月 8 日

引子

我们一般不写书。我们更喜欢把时间花在战场上，对客户的开发过程和组织进行优化、调试、重构。但最近留意到一个很明显的趋势，于是有写一点东西的必要了。给你看一个典型案例：

- **Jim:** “我们已经搞完 Scrum 了！”
- **Fred:** “效果怎么样？”
- **Jim:** “跟原先比起来好很多啊……”
- **Fred:** “嗯，但是呢？”
- **Jim:** “但是你也知道，我们是一个支撑维护团队。”
- **Fred:** “嗯，接着说。”
- **Jim:** “唔，我们喜欢这些东西，像在产品 backlog 里面排优先级啊，自组织团队啊，每日立会啊，回顾啊之类的……”
- **Fred:** “那问题在哪儿呢？”
- **Jim:** “我们的 sprint 一直失败。”
- **Fred:** “为什么呢？”
- **Jim:** “因为我们很难承诺出两周的计划。迭代对我们来说没太大意义，我们得看今天有啥活最急就干啥。是不是我们的迭代得改成 1 周？”
- **Fred:** “你能承诺 1 周的工作吗？你能全神贯注地、安静地工作一周么？”
- **Jim:** “恐怕不能，我们每天都有问题过来。也许得搞 1 天的 sprint……”
- **Fred:** “你们的问题在一天里面能解决么？”
- **Jim:** “不能，一般都得好几天。”
- **Fred:** “所以 1 天的 sprint 也不行啊。你考虑过完全放弃 sprint 么？”
- **Jim:** “不怕告诉你，我们还挺想的。不就是担心跟 Scrum 冲突么。”
- **Fred:** “Scrum 只是个工具。你可以选择什么时候用，什么时候不用。别成了工具的奴隶！”
- **Jim:** “那我们该咋办？”

- **Fred:** “你听过‘看板’吗？”
- **Jim:** “那是啥？它跟 Scrum 有啥区别？”
- **Fred:** “给，看看这本书吧！”
- **Jim:** “不过我倒是真得喜欢 Scrum 的其他部分，我现在就得换成看板吗？”
- **Fred:** “不用，你可以混着用。”
- **Jim:** “真的假的？咋整？”
- **Fred:** “接着看吧……”

本书的目的

如果你对敏捷软件开发感兴趣，那差不多就该听过 Scrum 了，也许还听过看板。我们最近越来越常听到这种问题，“什么是看板，它跟 Scrum 比起来怎么样？”它们各自都有哪些互补优势，会不会有潜在的冲突？

本书的目的就是帮你澄清迷雾，让你能够在身处的环境中给看板和 Scrum 找到合适的位置。

如果你觉得有收获的话，请告诉我们！

InfoQ
Enterprise Software Development Community

时刻关注企业软件开发领域的变化与创新

Java社区：企业Java社区的变化与创新

.NET社区：.NET相关的企业软件开发解决方案

Ruby社区：面向Web和企业开发的Ruby，主要关注Ruby和Rails

SOA社区：关于企业内面向服务架构的一切

敏捷社区：面向敏捷软件开发的项目经理

架构社区：设计、技术趋势及架构师所感兴趣的话题

运维社区：关注IT运维中的存储、监控和设计

官方微博：http://weibo.com/infoqchina
豆瓣小组：http://www.douban.com/group/infoq
讨论组：groups.google.com/group/infoqchina
编辑电邮：editors@cn.infoq.com
商务合作：sales@cn.infoq.com

第一部分——二者对比

本书的第一部分从实践角度出发，力求对 Scrum 和看板做出客观的对比。它实际上是对 2009 年 4 月发表的“看板 vs. Scrum”一文稍稍做了升级。那篇文章很受欢迎，所以我决定增补成一本书，邀请同事 Mattias 充实进一位客户 “在硝烟中”的故事，作为案例分析。这部分特别有料！你大可以跳过第一部分，从实战复盘开始阅读，我不会难过的。唔，也许会有一点点。

/Henrik Kniberg

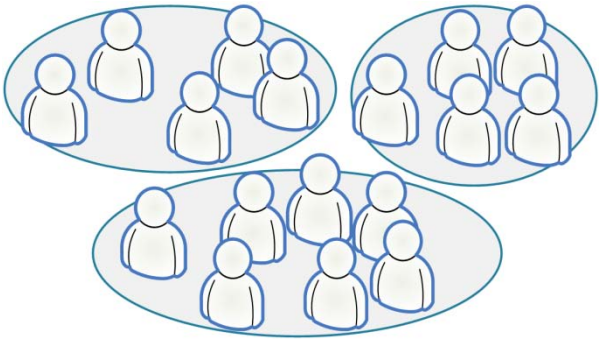
1

究竟什么是 Scrum，什么是看板？

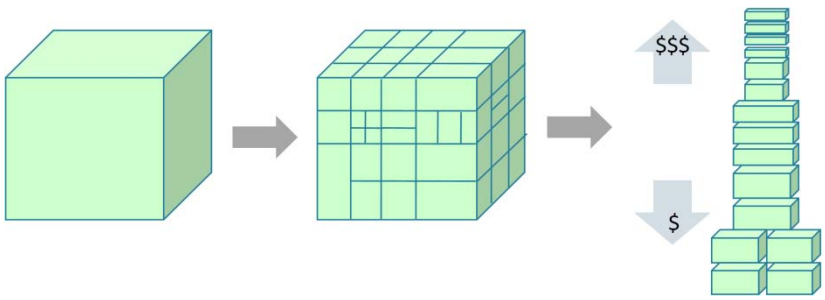
先分别用两百字以内的篇幅总结一下 Scrum 和看板吧。

Scrum 简述

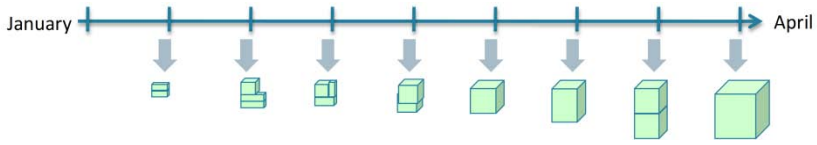
- **把组织拆分**成小规模的、跨功能的自组织团队。



- **把工作拆分**成一系列小而具体的交付物。按优先级排序，估算每项任务的相对工作量。



- **把时间拆分**成固定大小的短迭代（通常为 1-4 周），在每个迭代结束时对基本可以交付的代码进行演示。



- 在每个迭代结束后跟客户一起检查发布目标，并据此**优化发布计划**，更新任务优先级。
- 每个迭代结束后进行回顾，进行**过程优化。**

我们不是靠一个**庞大的团队**，花**大量时间**造出**庞然大物**；而是用**小团队**在**短时间**内做出**小块的东西**来，在**有规律的集成**中组装出全貌。

203 个字……差不多行了。

更多细节可以参考“硝烟中的 Scrum 和 XP”。这本书有免费的在线版本。我认识它的作者，他是个好人:o)

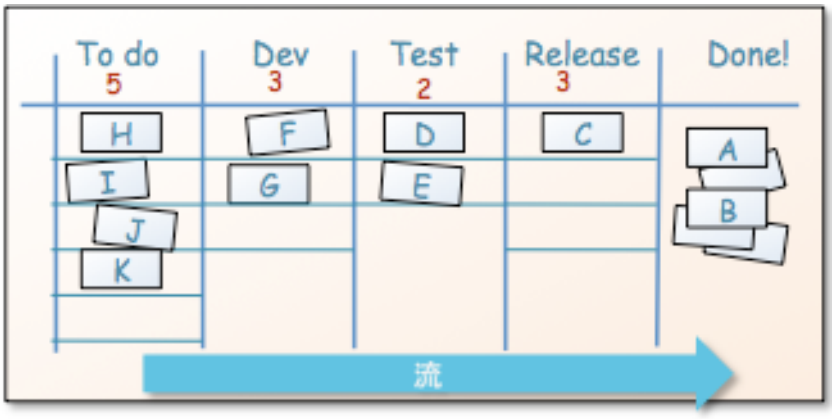
http://www.crisp.se/ScrumAndXpFromTheTrenches.html

（译者注：该书也有中文电子版免费下载，请参见下述链接：http://www.infoq.com/cn/minibooks/scrum-xp-from-the-trenches ，我认识这本书的译者，他也是个好人^_^）

在 http://www.crisp.se/scrum 上面还有更多 Scrum 相关的链接。

看板简述

- **将流程可视化**
  - 把工作拆分成小块，一张卡片写一件任务，再把卡片放到墙上。
  - 每一列都起一个名字，显示每件任务在流程中处于什么位置。
- **限制 WIP**（在制品，work in progress）——明确限制流程中每个状态上最多同时进行的任务数。
- **度量生产周期**（完成一件任务的平均时间，又称循环周期），对流程进行调优，尽可能缩短生产周期，并使其可预测。



我们收集了一些跟看板相关的链接，很有用。它们放在： http://www.crisp.se/kanban

2

Scrum 和看板有什么关系？

Scrum 和看板都是过程工具

工具=用于完成任务或达成目的的任何东西

过程=工作方式

Scrum 和看板都是过程工具，它们讲的是做哪些事情能够在一定程度上帮助你提高工作效率。Java 也是工具，它让编程更加简单。牙刷也是工具，它让你够得到牙齿，方便清洁。

比较是为了更好地理解，而不是评判优劣

刀叉哪样更好？

这个问题很没意思，不是么？上下文不一样，答案也就不一样。吃肉丸最好用叉子；切蘑菇最好用刀子；敲桌子用哪个都行；吃牛排就得同时上阵；吃米饭的话……唔……有人喜欢用叉子，有人就更喜欢用筷子。

所以在比较工具的时候得谨慎一些。比较是为了更好地理解，而不是评判优劣。

工具都不是全面的，也都不是完美的

跟所有的工具一样，Scrum 和看板既不完美，也不全面。它们没有把需要做的事情全都告诉你，只是给了一些明确的约束和指导。比如说，Scrum 的约束是固定时长的迭代和跨功能团队，看板的约束是要有可见的板，队列大小要有限制。

有意思的是，工具的价值恰恰在于*它限制了你的选择*。如果有一款过程工具，让你什么事情都能做，那它就没多大用了。我们管这种工具叫做“做啥都行”，或者美其名曰“做正确的事”。“做正确的事”这个过程肯定管用。它就是银弹！要是没生效，那肯定是因为你没按照这个过程工作 :o)

使用恰当的工具可以帮助你成功，但不能确保成功。人们很容易把*项目*成败跟*工具*成败弄混。

- 有一款伟大的工具，项目可能成功。
- 有一款差劲的工具，项目可能成功。
- 有一款差劲的工具，项目可能失败。
- 有一款伟大的工具，项目可能失败。

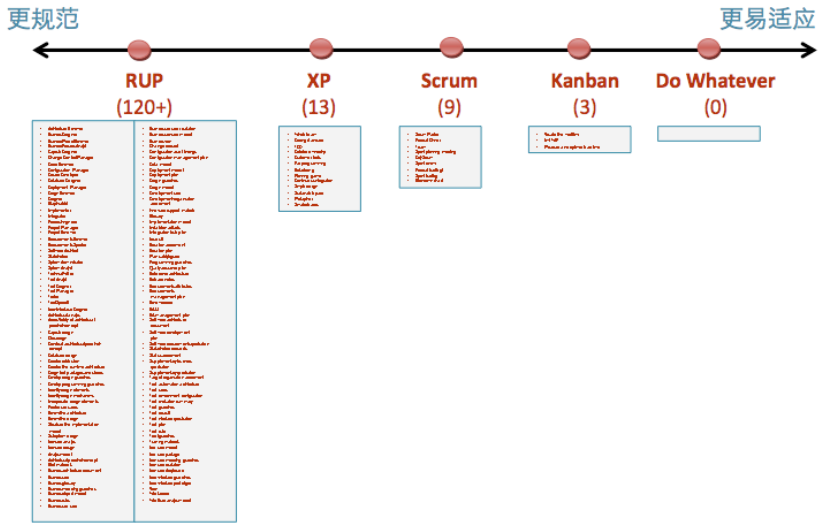
Scrum 比看板更规范

我们可以从每个工具都有哪些规则的角度来进行比较。**规范性**指的是“要遵守更多的规则”，**适应性**指的是“要遵守的规则较少”。在 100%规范性的情况下，你就根本不用动脑子了 ,不管做什么事情都按规则行事 ;而 100%的适应性就意味着**做什么都行**，没有任何规则约束。你肯定能看出来，这两种极端都是很荒谬的。

敏捷方法有时被称作*轻量级*方法,主要原因就在于它们不如传统方法那么规范。敏捷宣言的第一条就是“个体和交互胜过过程和工具”。

Scrum 和看板都是适应性很强的，但相对而言，Scrum 更规范一些。Scrum 多了些约束，少了些选择。比如 Scrum 要求使用有固定时长的迭代，但看板没有。

下面从规范性和适应性的角度来比较一些工具：



RUP 的规范性相当强——它有 30 多种角色，20 多种活动，70 多种工件；需要学习的东西不胜枚举。当然，你肯定不会全都用上，为自己的项目选一个合适的子集就行了。但不幸的是，操作起来可是颇有难度。“唔，我们会需要*配置审计决定*这个工件么？我们会需要*变更控制经理*的角色么？还定不下来啊，最好还是先留着吧。”这可能就是为什么 RUP 比起敏捷方法来——例如 Scrum 和 XP——用到最后往往令人不堪重负了。

XP（极限编程）比 Scrum 又规范一些。它囊括了 Scrum 的大部分内容，还多了很多相当具体的工程实践，例如测试驱动开发和结对编程。

Scrum 的规范性比 XP 弱 ,因为它没有规定任何具体的工程实践。但它又比看板规范，因为它规定了迭代和跨功能团队之类的东西。

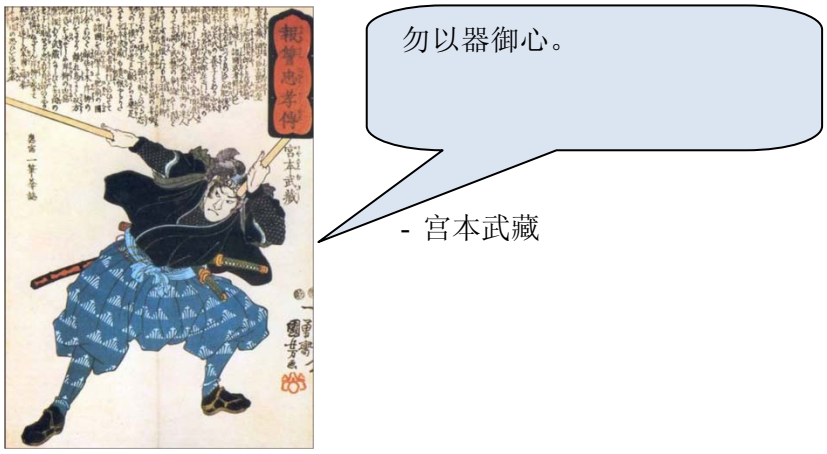
Scrum 和 RUP 的主要区别在于，RUP 给你的东西太多了，你得自己把不需要的东西去掉；而 Scrum 给你的东西太少了，你得自己把需要的东西加进来。

看板几乎对任何做法都是开放的。它仅有的约束就是将**流程可视化**和**限制在制品**。它离**做什么都行**只有几步之遥，但仍有令人惊异的力量。

别把自己绑在一种工具上！

把工具搭配着用，用在合适的地方！我无法想象几乎不用 XP 的 Scrum 团队还能成功。很多看板团队也在做每日立会（Scrum 实践）。有些 Scrum 团队也把 backlog 条目写成**用例**（RUP 实践），还会限制队列大小（看板实践）。只要有用就行。

宫本武藏曾说过（17 世纪的著名武士，以二刀流闻名于世）：



不过我们还是要关注每样工具有哪些约束的。假如你在用 Scrum，又决定不用固定时长的迭代（或是其他任一款 Scrum 的要素），就不要说你在用 Scrum 了。Scrum 本身已经足够浓缩了，如果你去掉一些东西，然后还叫它 Scrum，那这个词就失去了意义，只会带来困扰。你可以给它起个别的名字，比如“Scrum 衍生品”，或者“Scrum 子集”，要不就“Scrum 似是而非”也行。:o)

3

Scrum 规定了角色

Scrum 规定了三种角色：产品负责人（描绘产品远景，定义优先级）、团队（实现产品）、Scrum Master（消除障碍，带领过程运作）。

看板没规定任何角色。

这可不是说你不能或是不应该在看板里有产品负责人的角色。这只是说你不是*非有不可*。不管是用看板还是 Scrum，你都可以根据需要增加任意角色。

但是增加角色的时候得小心，要确保新增的角色能带来价值，而且不要跟其他方面有冲突。你觉得你真的需要项目经理么？在大项目里面也许挺好，因为这个人可以在多个团队和多个产品负责人之间进行协调。但在小团队里面这个角色也许就是浪费，甚至更糟——导致局部优化和微观管理。

Scrum 和看板有一个共同的思路："少就是多"。有疑虑的时候，先从少做起。

在后续章节中，我会用"产品负责人"这个词来表示能够给团队设置优先级的人，不管用的是什么过程。

4

Scrum 规定了固定时长的迭代

固定时长的迭代是 Scrum 的基础。你可以选择迭代长度，但一般都会在一定时间内让迭代长度固定不变，继而形成节奏。

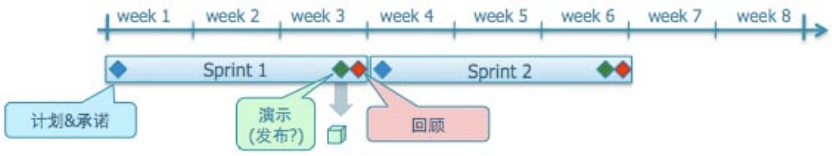
- **迭代伊始**：综合考虑产品负责人定义的优先级和自己的生产率，团队从产品 backlog 里面挑选出一定数量的条目，创建迭代计划。
- **迭代进行中**：团队全心投入所承诺的任务。迭代范围已固定。
- **迭代结尾**：团队向相关干系人演示他们可以工作的代码，理想情况下，这些代码基本上是可以发布的（经过测试可以交付）。然后团队进行回顾，讨论如何改进过程。

所以 Scrum 的迭代就是一段长度固定的单声部旋律，混合了三种活动：计划、过程改进、（理想中的）发布。

看板没有规定固定时长的迭代。你可以选择什么时候做计划，什么时候改进过程，什么时候发布。你还可以选择是有规律的采取行动（如每周一发布），还是按实际需要进行（如有了有用的东西之后就发布）。

团队 1（单声部）

“我们用了 Scrum 的迭代”



团队 2（三声部）

“我们有三个声部。每周都把能发布的东西发布出去。每两周开一次计划会议，更新优先级和发布计划。每四周做一次回顾，调整改进过程。”

团队 3（几乎是事件驱动）

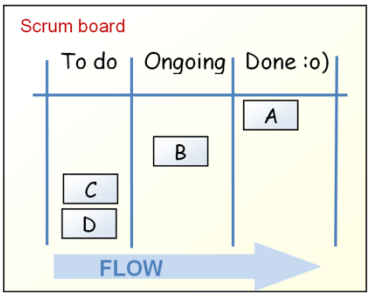
“我们在快没事情做的时候开计划会议。有 MMF（最小适销特性，minimum marketable feature）能发布的时候就发布。当我们第二次遭遇同样问题的时候就自发组建质量圈。每四周还进行更加深入的回顾。”

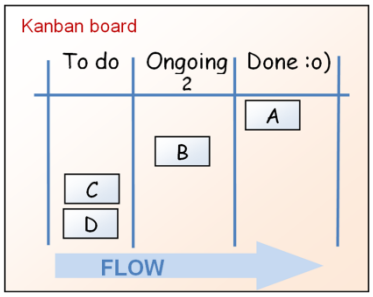
5

看板按流程状态限制 WIP，Scrum 按迭代限制 WIP

Scrum 的 Sprint backlog 显示了当前迭代（迭代也就是 Scrum 术语中的 Sprint）要完成哪些任务。它们一般都用墙上的卡片展示，被称作 Scrum 板（Scrum board），或是任务板（Task board）。

那 Scrum 板和看板图(Kanban board)有什么区别呢？用个简单的小项目比较一下：





在两个案例中，我们都追踪了几项任务在流程中的进展。我们选了三个状态：To Do，Ongoing，Done。你自己想用什么状态都行，比如有的团队还增加了 Integrate、Test、Release 等状态。不过，这个时候不要忘了*少就是多*这个原则哦。

那这两块样板的区别是什么呢？喏——就是看板图中间那一列上的那个小字 2 啊。就是那点东西。2 的意思是“不管什么时候，这一列上最多有两个任务”。

换成 Scrum 的话，团队大可以把所有东西都放到 Ongoing 那一列里面去！但因为迭代本身的范围是固定的，所以 Scrum 依然有个潜藏的限制。这里的潜在限制就是每列最多放 4 张卡，因为整个板上也就只有 4 张。看板直接限制了 WIP，Scrum 是间接限制的。

大多数 Scrum 团队最终都会意识到，有太多进行中的任务不是什么好事，然后慢慢形成文化，在做新东西之前先把现有的做完。有的甚至会决定明确限制 Ongoing 这一列里能放多少卡片——吼吼！——Scrum 板就这样变成看板图了！

所以，Scrum 和看板都是限制 WIP 的，只是方式不同。Scrum 团队通常都要度量*生产率*——每个迭代能完成多少条任务（或是用相应的单位表示，如“故事点”）。一旦他们知道了自己的生产率，这个数值就成了 WIP 的上限（或者至少是个参考值）。平均生产率是 10 的团队一般不会在一个 Sprint 里面放进超过 10 张卡(或故事点)。

Scrum 的 WIP 按**单位时间限制**。

看板的 WIP 按**流程状态限制**。

在上面的看板图中，不管旋律有多长，在任何时间点上，“Ongoing”这个状态上最多只能有两张卡。你自己可以决定给哪个状态设置多大上限，但一般而言，整条价值流上所有的状态都要加上上限，开始点越早越好，结束点越晚越好。所以再看前面的例子，我们应该考虑把“To Do”（不管它叫什么，总之是输入队列的起点）也加上 WIP 上限。一旦 WIP 上限都设置完毕，我们就可以开始度量和预估生产周期——即一张卡片在图上从头移动到尾所用的平均时间。估好了生产周期，我们就能对 SLA（服务品质协议，service level agreements）做出承诺，制定出合理的发布计划。

如果各项任务的规模差异很大，那可能就得考虑用故事点或是其他尺寸单位来定义 WIP 上限了。有些团队会花时间把任务都拆分得大小差不多，从而避免费心考虑用什么单位限制 WIP，也减少了估算的时间（你或许会认为估算也是浪费）。任务大小相近，就更容易创建一个平滑流动的系统了。

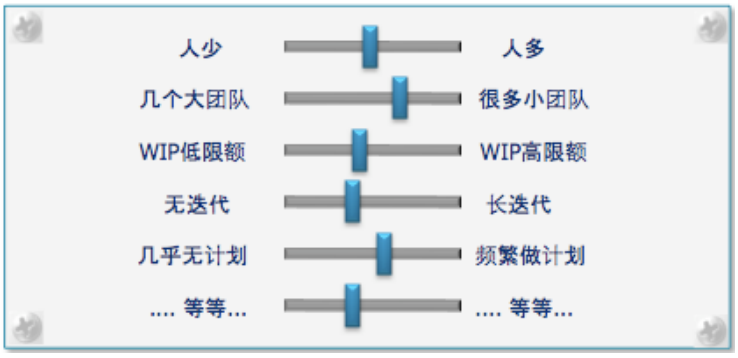
6

二者都是经验主义的

假如在这些仪表上有旋钮，拧一拧就能把过程配置好。“我想要高生产力，短生产周期，高质量，高预测性。我就把它们拧到 10、1、10、10。”

这不是很美妙么？可惜这些能直接控制的东西真没有。至少我不知道哪里有。如果你知道的话请告诉我。

我们倒是有一些可以间接控制的东西。



Scrum 和看板都是经验主义的产物，你用的时候需要先进行试验，然后根据自己的环境作调整。实际上，你*必须*得先试验。Scrum 和看板都没给出一切问题的答案，它们只是给了一些基本约束，以此驱动过程改进。

- Scrum 说，你应该有跨功能团队。那团队里面应该有什么人？不知道，自己试吧。
- Scrum 说，团队选择 Sprint 里面放多少东西。那到底要放多少东西？不知道，自己试吧。
- 看板说，你应该限制 WIP。那到底上限是多少？不知道，自己试吧。

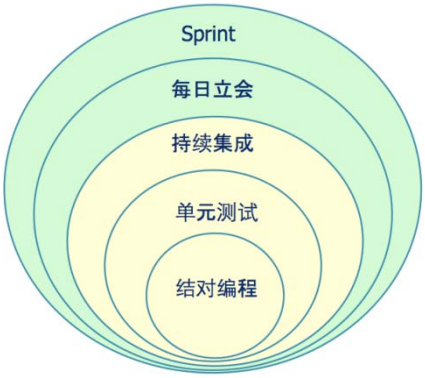
我前面提到过，看板的约束比 Scrum 少。这样一来，你就得要考虑更多因素，有更多旋钮要调。这种方式的利弊很难一句话说清楚，要看具体环境。假设你要配置一款软件工具，打开对话框以后，你希望看到 3 个选项还是 100 个选项？也许是在 3 ~ 100 之间的某个数吧。这得看你想要调整多少东西，你对这个工具的熟悉情况如何。

我们猜想减少 WIP 上限可以改善过程，于是就这样做了。与此同时，也在观察生产率、生产周期、质量、可预测性等数据的变化。从观测结果中得出结论，再多做一些调整，于是我们的过程就在持续改进。

这种方式有很多叫法：改善（Kaizen。即持续改进，精益术语），内省与调整（Inspect & Adapt，Scrum 术语），经验式过程控制（Empirical Process Control），乃至科学方法（The Scientific Method）。

它最核心的一点就是*反馈环*。改变 => 检查结果 => 从中学习 => 继续改变。一般而言，反馈环越短越好，这样可以快速调整过程。

Scrum 的基本反馈环就是 Sprint，跟 XP（极限编程）合并的话就会再多几个：



做法得当的话，Scrum+XP 可以提供不少非常有用的反馈环。

最里面的反馈环——结对编程，它的反馈周期只有几秒钟。缺陷产生以后，分分秒秒就能被找到并解决（嘿，那个变量不应该是 3 么？）。这个反馈周期的意义是回答：我们做的结果正确么？

最外圈的反馈环——Sprint，它的反馈周期有几周。它的意义是回答：我们定的目标正确么？

那看板的反馈环在哪里呢？唔，首先一点，不管你用不用看板，你都可以（或者应该）把上面的反馈环全都用上。然后参考看板给你的几个很有用的实时度量指标。

- 平均生产周期。每次有任务到达“Done”这一列（不管它叫什么吧，反正是最右边那一列）的时候就更新数据。
- 瓶颈。典型症状就是 X 列里面堆满了卡片，但是 X+1 列里空空如也。找找板上哪里有“气泡”吧。

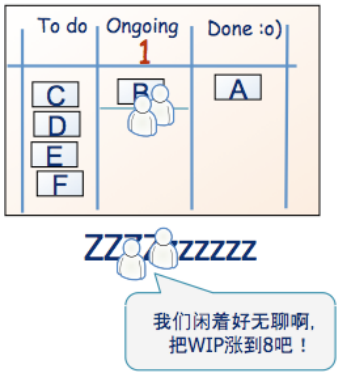
用实时度量指标的一个好处就是，你可以根据自己想要分析指标、调整过程的频率，来选择反馈环的长度。太长的反馈环会导致过程改进速度过缓。太短的反馈环会导致过程变化太快，没有时间稳定，白做无用功。

实际上，反馈环的长度本身也是需要实践调整的……这个过程可以称作反馈环的反馈环。好了，我还是打住吧。

例子：看板中的 WIP 上限试验

WIP 上限是看板的典型“调整点”之一。我们怎么知道配置得好不好呢？

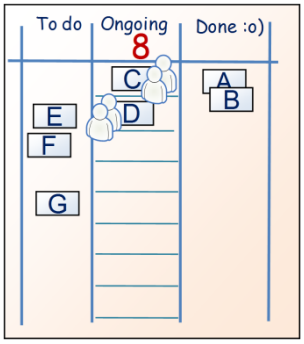
假设我们有一个 4 人团队，决定一开始把 WIP 限定为 1。



不管什么时候开始做一项任务，它做完之前都不能做新的。所以它很快就能完事。

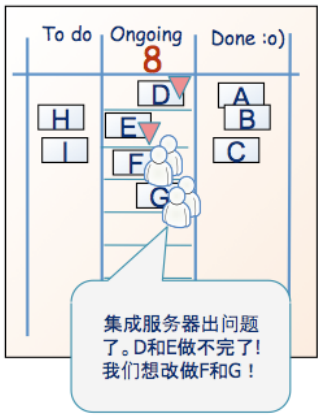
这多好啊！但 4 个人都一拥而上做同一张卡片一般不太可行（在上面的例子里），所以就有人发呆了。如果这种情况只是偶尔发生还不碍事，要是常常出现的话，平均生产周期就会变长了。从根本上说，WIP 设成 1 以后，任务在“Ongoing”状态只会停留不长时间，但它们会在“To Do”状态上大量阻塞，所以整个流程的总生产周期要比理想情况下长很多。

如果 1 太小的话，把 WIP 改成 8 怎么样？

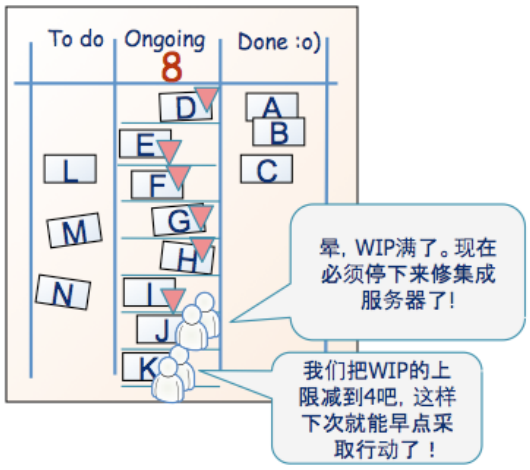


在一段时间内这还挺有效果的。我们发现结对工作的时候效率最高，所以 4 人团队的话，一般在任何时刻都有两个进行中的任务。8 只是 WIP 的上限，实际没达到那么多也挺好！

但现在想象一下，如果我们的集成服务器出问题了，哪件事都没法彻底做完（我们的“完成”定义中包括了集成）。这种事是难免的，不是吗？

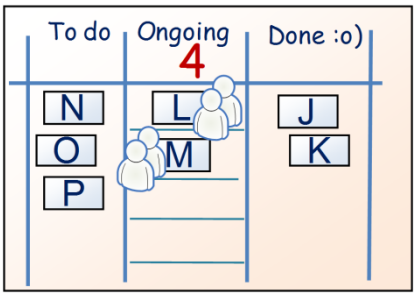


D 和 E 完不了了，我们就改做 F。但 F 也集成不了，就把 G 也拉进来一起做。搞来搞去，看板就到上限了——“Ongoing”里面挤了 8 张卡。



到了这份上，一张卡也拉不进来了。集成服务器不修不行了！WIP 上限提醒我们要采取行动，改善瓶颈；而不是把没完成的工作堆啊堆啊堆啊堆个没完。

这样挺好。但要是 WIP 当初设成 4 的话，我们早就能有所反应了，这样平均生产周期还能表现得好点。所以这是要平衡的。我们度量平均生产周期，不断优化 WIP 上限，以此优化总生产周期。



再过上一阵子，可能又会有卡片在“To Do”那里堆起来，这时候可能又需要增加 WIP 上限了。

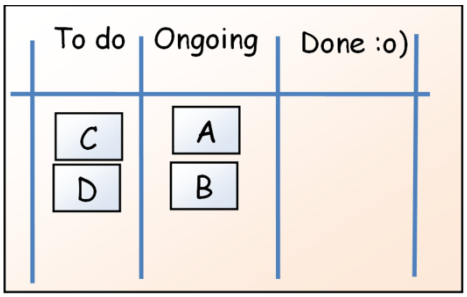
我们到底为什么需要一个“To Do”呢？唔，要是不管什么时候团队去找客户问需求，客户都有空，那就不需要“To Do”了。但有时候客户没空，所以“To Do”这一列是给了团队一个小缓冲，让他们在这种时候能有东西拉过来做。

要尝试！或者换成 Scrum 学家的说法，自省与调整！

7

Scrum 在迭代内拒绝变化

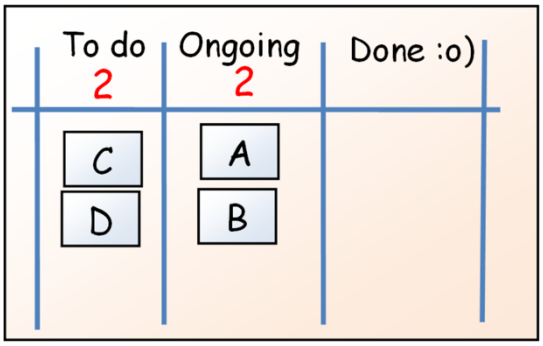
假设我们有一块这样的 Scrum 板



如果有人跑出来要把 E 放上去会怎么样？

Scrum 团队一般会说，“对不起，这样不行。我们已经承诺了这个 sprint 要做完 A+B+C+D。不过你可以把它放到产品 backlog 里面。如果产品负责人觉得它优先级很高的话，我们会从下一个 sprint 开始做。”长度适中的 sprint 能够让团队有足够的时间全心全意把一些事情做完，而与此同时，产品负责人依然可以有规律地调整优先级。

要是换成看板团队，他们会怎么说？



他们会说“请把 E 放到 To Do 那一列上。但那一列的限额是 2，所以你得把 C 或者 D 去掉。我们现在在做 A 和 B，只要腾出手来，我们就会把 To Do 顶上的卡片拉过来做。”

看板的原则是“一件出去，一件进来”（由 WIP 驱动），所以看板团队的响应时间（多久才能响应优先级的变化）就等于他们要花多长时间才能把手头的事情做完。

Scrum 的平均响应时间等于 sprint 长度的一半。

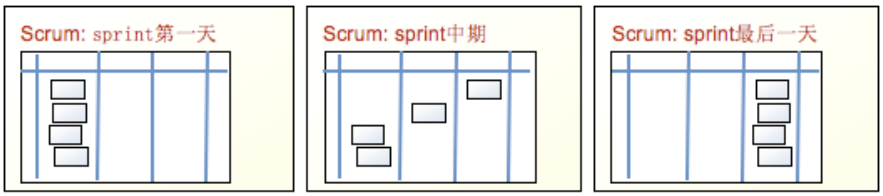
用 Scrum 的时候，产品负责人不能碰 Scrum 板，因为团队已经对迭代中要完成的一些具体任务做了承诺。而用看板的时候，你得自己规定哪些人能动板子。一般来讲，产品负责人可以操作的都是最左边的几列，比如“To Do”、“Ready”、“Backlog”、“Proposed”。这几列他什么时候改都行。

但这两种方式不是互相排斥的。Scrum 团队*也可以*允许产品负责人在 sprint 中期更改优先级（虽然这往往都会被当作异常情况）。看板团队*也可以*对修改优先级的时机做限制。看板团队甚至可以使用固定期限固定承诺的迭代，就跟 Scrum 那样。

8

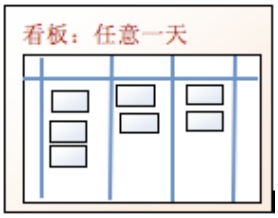
Scrum 板在迭代之间重置

在 Sprint 的不同时期，Scrum 板通常有不同的样子：



Sprint 结束以后，板子就会进行清理——所有卡片全部去掉。等到新的 Sprint 开始，Sprint 计划会议结束以后，我们就有了新的 Scrum 板，最左边的一列上也有了新卡片。从技术上讲这也是浪费，但是有经验的 Scrum 团队不会花太长时间在这上面，而且重置板子的过程会给人带来美妙的成就感和满足感。这就跟吃完饭刷锅碗瓢盆一样，过程很痛苦，但刷完以后看着干干净净的碗碟就会心情舒畅。

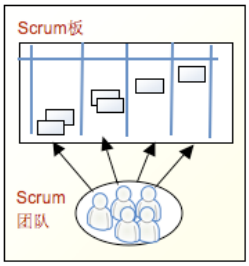
看板图的样子几乎是一成不变的——你不需要把板子清理干净，重新开始。



9

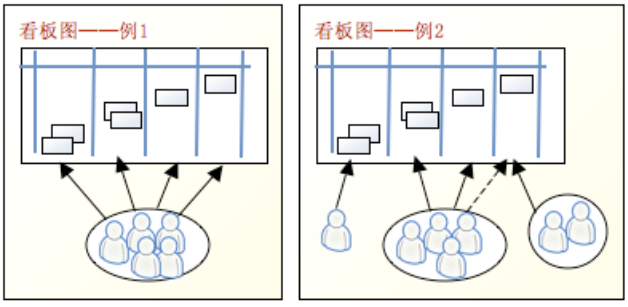
Scrum 规定了跨功能团队

一个 Scrum 团队只有一块 Scrum 板。Scrum 团队是跨功能的，要完成迭代全部任务所需的技能，这个团队要全都具备。Scrum 板对所有感兴趣的人全都是可见的，但只有它的所属 Scrum 团队才能编辑——这是他们管理迭代承诺的工具。



看板不强制要求跨功能团队，看板图也不是独归某个团队所有。看板图对应的是流程，不必非得是一个团队。

下面是两个例子：



例 1：整块板子为一个跨功能团队使用。就跟 Scrum 一样。

例 2：产品负责人设置第一列的优先级。一个跨功能开发团队做开发（第二列）和测试（第三列）。有个专门的团队做发布（第四列）。他们的能力略有重叠，所以如果发布团队成了瓶颈，就可以让开发人员过去帮忙。

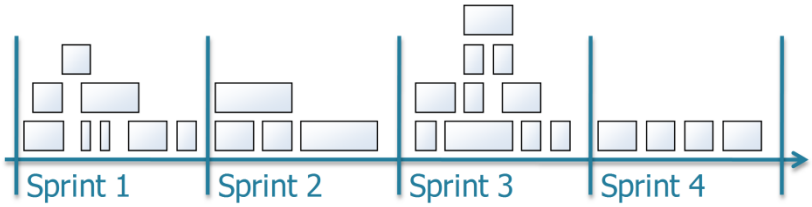
所以，用看板的时候，你需要建立一些基准，规定哪些人可以用看板，怎么用看板，用这些基准进行试验，优化流程。

10

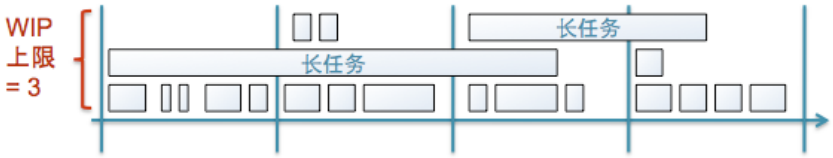
Scrum 的 backlog 条目必须能跟 Sprint 搭配的上

Scrum 和看板都以增量开发为基础，即将工作拆分成小块。

Scrum 团队只会承诺他们认为能在一个迭代里面做完（基于他们对“完成”的定义）的任务。如果任务太大了，一个 Sprint 放不下，团队跟产品负责人就会寻找方法拆分，直到能放下为止。如果任务都比较大，迭代就会较长（虽然一般都不会超过四周）。



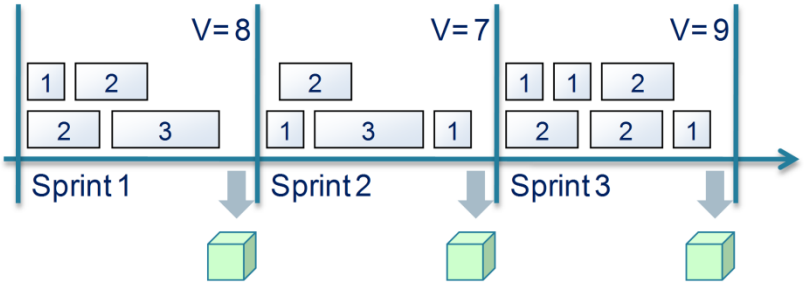
看板团队努力缩短生产周期，保持顺畅流动，而这些因素会间接推动团队把任务拆分成相对较小的片段。但是看板对任务规模没有明文规定一定要在某个时间内做完。在同一张板上，我们可能会既有 1 个月做完的卡片，又有一天能做完的卡片。



11

Scrum 规定了估算和生产率

在 Scrum 里面，团队要对每个承诺的任务估算其相对大小（=工作量），到迭代结束的时候，把每个任务的大小相加，就得到了生产率。生产率是度量团队能力——我们每个 Sprint 能交付多少东西——的指标。下面是平均生产率为 8 的一个例子。



知道平均生产率为 8 是件好事，因为我们可以据此合理推测接下来的 Sprint 能做完多少任务，继而做出合理的发布计划。

看板没有规定估算这回事。所以如果需要作承诺的话，就得想一想怎么作预测了。

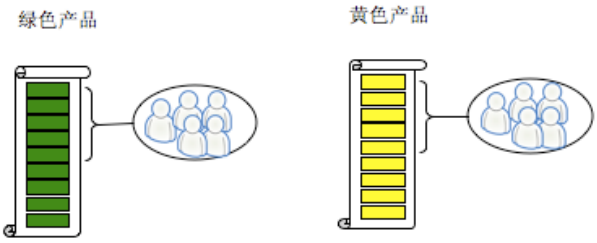
有的团队用 Scrum 的方式作估算，度量生产率。有的团队就跳过估算，但是尽力把每个任务都拆分得大小相近——于是生产率就很好算了，把单位时间里做完的任务数加起来就行（例如每周完成的特性数）。有的团队会把任务打包成 MMF（最小适销特性），然后度量平均每个 MMF 的生产周期，以此建立 SLA（服务品质承诺）——例如“我们承诺的单个 MMF 一般都会在 15 天之内交付”。

以看板风格制定发布计划和承诺管理的方式有很多，也很有趣。但是看板没有规定任何具体的方式，所以就去 Google 吧，多试几种不同的做法，直到找到最适合你的为止。也许我们有一天会看到一些“最佳实践”的。

12

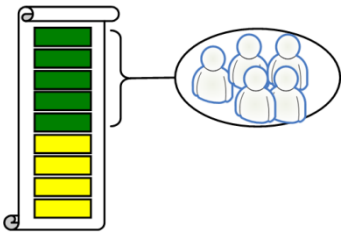
二者都允许在多个产品上并行工作

“产品 backlog”这个词其实挺不幸的，因为听上去就像是所有任务都得是跟同一个产品相关的。下面是两个产品，绿色和黄色，每一个都有自己的产品 backlog，也有自己的团队。

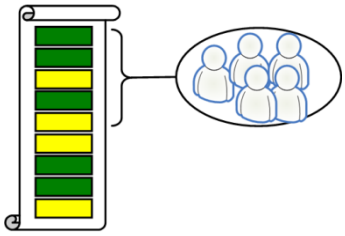


万一你只有一个团队呢？其实，你可以把产品 backlog 当作团队 backlog 看待。如果团队在维护多个产品，就把这些产品都合并到一个列表里面。这会迫使我们把产品放在一起考虑优先级，有时候会给我们帮上很大忙。我们可以采取多种做法：

其一，每个 Sprint 聚焦一个产品：

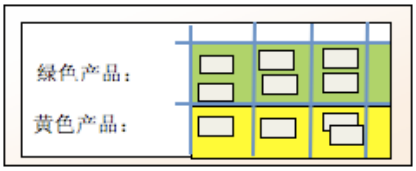


其二，每个 Sprint 里面，两个产品的特性都要做：



后者跟看板是一样的。在一张图上可以有多个产品流动。我们可以用不同颜色的卡加以区分：

也可以用“泳道”来区分：



13

二者都是既精益又敏捷

一般来讲，Scrum 和看板跟精益思想和敏捷宣言里面的价值观和原则是相当吻合的，当然我就不在这里过一遍了。举些例子看看：

- Scrum 和看板都是拉动式计划系统，跟精益的 JIT（准时化）库存管理原则是一致的。这表示团队决定从什么时候开始干活，干多少活。等他们准备就绪的时候就把工作“拉”过去，而不是从外部“推”进来。就像打印机只有在准备打印的时候才把下一页拉进去一样（当然，打印机还是有些小小库存的）。
- Scrum 和看板都基于持续的、经验主义的过程优化，这跟精益的改善原则是一致的。
- Scrum 和看板都强调响应变化胜过遵循计划（虽然看板的响应速度一般要比 Scrum 快），这是敏捷宣言的四项价值观之一。

……

从某个角度来看，Scrum 也不那么精益，因为它把批量任务放到固定期限的迭代里面去。但这也要看迭代有多长，还要看跟谁比。用传统过程的时候，我们一年也许会集成并发布 2 ~ 4 次，这样来看的话，每两周就能生产出可交付代码的 Scrum 可就是精益无比了。

但是，如果你不断缩短迭代周期，本质上等于向看板靠拢。如果你们开始研究把迭代缩成比 1 星期还短，那还不如干脆把固定期限的迭代干掉了。

有句话我前面提到过，接下来还会不断重复：一直尝试，直到找到适合你的方式为止！然后继续尝试:o)

14

小小差异

Scrum 和看板的区别已经讲过一些了，还有些区别不是那么鲜明，但也值得一看。

Scrum 规定了经过优先级排序的产品 backlog

Scrum 的优先级是通过产品 backlog 排序体现的，优先级的变化会在下一个（不是当前的）Sprint 生效。在看板里面，你可以选择任何一种定义优先级的方式（或者干脆没有），一旦有人腾出空来，优先级的变化就可以生效（不像 Scrum 要有固定的时间点）。看板可以有也可以没有产品 backlog，即便有的话，排不排优先级也不一定。

但操作上的差异并不大。看板图上最左边那列的用途跟 Scrum 产品 backlog 基本相同。不管这列排不排优先级，团队总要做出某种决定，先拉哪张卡片来做。比如：

- 总是选最上面那张。
- 总是选最早的那张（所以每张卡都有时间戳）。
- 随便选。
- 花大概 20%的时间做维护的任务，80%的时间做新特性。
- 把团队的生产力平均分配到产品 A 和产品 B 上。
- 如果有红色卡片就先选红的。

Scrum 的产品 backlog 也可以用成看板。我们可以限制它的大小 定义怎么排优先级。

Scrum 规定了每日立会

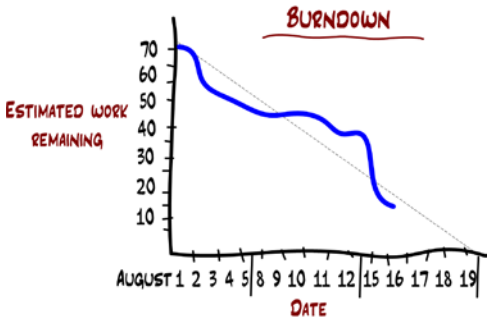
Scrum 团队每天都会在同一时间同一地点开一个短会（最多 15 分钟）。这个会议的目的是让大家知道工作进展，计划当天任务，识别严重问题。它有时被称作每日立会，因为一般都是站着开的（站着开时间短，大家也更有精神）。

看板没有规定每日立会，但绝大多数看板团队好像也都在这样做。这是项伟大的技术，不管你用什么过程都值得采用。

在 Scrum 里面，会议形式是以人为中心的——每个人轮流发言。很多看板团队的会议重心向图倾斜，重点放在瓶颈和其他可见的问题上。这种方式的扩展性更高。如果有共用一张图的 4 个团队一起开每日立会，也许就不必让每个人都发言，只要聚焦瓶颈就行了。

Scrum 规定了燃尽图

Sprint 燃尽图每天更新，展示当前迭代还剩多少工作没做。



Y 轴的单位跟 Sprint 任务的单位一样，一般都是小时或天（如果把 backlog 条目进一步拆分成任务的话），也或者是故事点（如果没拆分的话）。选择其实还蛮多的。

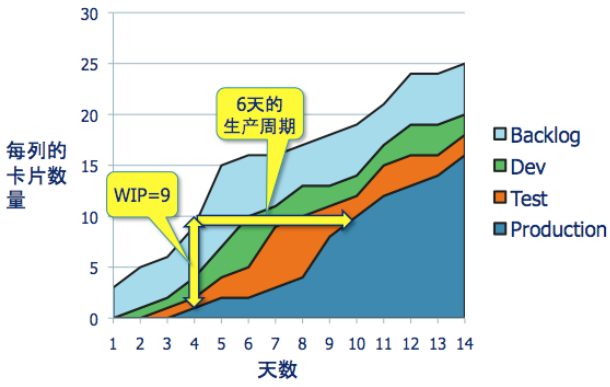
Scrum 把 Sprint 燃尽图用作跟踪迭代进度的主要工具。

有些团队还用了发布燃尽图，格式跟 Sprint 燃尽图一样，只是层次不同——它通常显示的是每过一个 Sprint，产品 backlog 上还剩多少故事点。

燃尽图的主要目的是便于尽早发现实际进度跟计划的偏差，好作出调整。

看板没有要求燃尽图。它根本没要求任何特殊的图表。但不管什么图表，只要他们想用，就肯定是可以用的（也包括燃尽图）。

下面是累积流图（Cumulative Flow diagram）的一个例子。它生动地展示出流动的平滑与否，WIP 如何影响生产周期。



看看它是怎么工作的。每天都有人算一下看板图上每列都有多少卡片，把数字沿 Y 轴方向堆起来。所以在第四天的时候，图上一共有 10 张卡片，从最右列看起，Production 有 1 张，Test 有 1 张，Dev 有 2 张，Backlog 有 6 张。（译者注：原文中所写的卡片数量有误，此处的修正经过了作者确认。）如果我们用每天的这些点连起来作图，就会看到上面这张漂亮图表了。横竖箭头显示了 WIP 和生产周期之间的关系。

横箭头告诉我们，在第 4 天加到 backlog 里面的卡片，平均要过 6 天才能到 Production 这一列，而且有将近一半时间处于 Test。图中可以看出，如果我们限制了 Test 和 Backlog 的 WIP，总生产周期就会大大缩短。

深蓝色区域的斜率代表了生产率（每天能够部署多少张卡）。我们会逐渐看到，高生产率会缩短生产周期，高 WIP 限额会延长生产周期。

大多数组织都想提高做事效率（=缩短生产周期）。但不幸的是，很多组织都落入陷阱，试图用增加人手或工作时间的方式来达成这个目的。一般而言，想提高做事效率，最有效的方式是让流动平滑起来，按能力限制工作数量，而不是加人或者让人工作得更辛苦。上面这种图显示出了背后的道理，会让团队和管理层更容易提高协作效率。

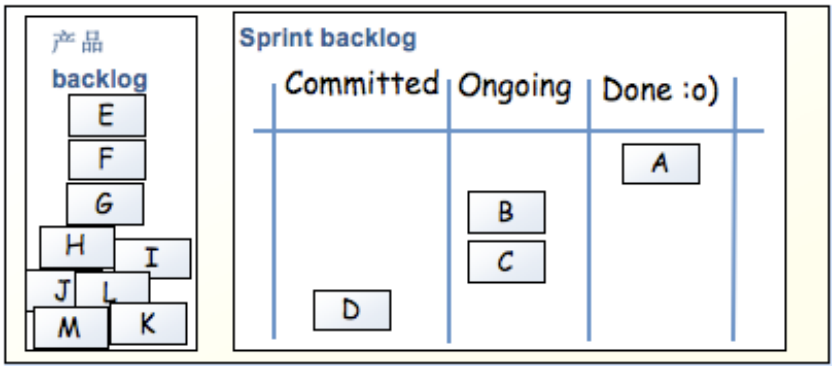
如果我们把排队状态（如“等待测试”）和工作状态（如“测试”）分开的话，道理就更明显了。我们想尽全力减少排队的卡片数，而一张累积流图可以提供恰当的契机。

15

Scrum 板 vs. 看板图—— 一个不大不小的例子

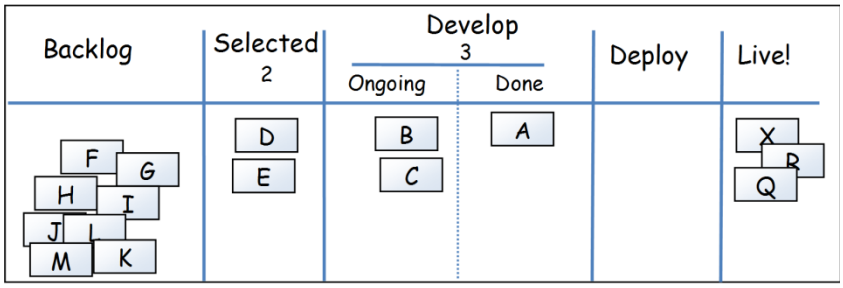
Sprint backlog 只是 Scrum 全景中的一部分而已——它显示出团队在当前的 Sprint 里面在做些什么。产品 backlog 也是一部分——这个列表装的就是产品负责人想在后面的 Sprint 里面做完的东西。

产品负责人可以看 Sprint backlog，但不能碰。他随时都可以改产品 backlog，但是他做出的修改直到下个 Sprint 才生效（生效=影响当前进行的工作）。



Sprint 完成以后，团队把“基本可上线的代码” 交付给产品负责人。然后 Sprint 就结束了，团队会做一个回顾，会自豪地向产品负责人演示特性 A、B、C、D。产品负责人会决定到底要不要让这些代码上线。最后一部分——真正把产品上线——通常不是 Sprint 要做的事情，所以也不会在 Sprint backlog 里面体现出来。

把上面的场景换成看板图来表示的话，大概是这样的：



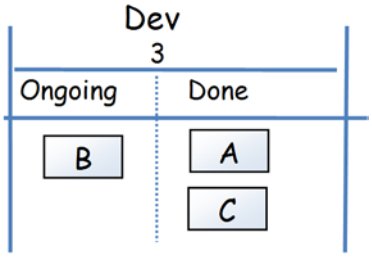
现在整个流程就都展现在同一张图上了——我们看到的不只是一个 Scrum 团队在一个迭代里面是怎么工作的。

在上面的例子中，“Backlog”这一列只是个总的愿望列表，没按特定顺序排列。“Selected”这列放的是优先级最高的需求，它的看板上限是 2。所以不管在什么时候，这列里面最多只能放两张卡片。一旦团队有人手能抽出来做新任务，他们就从“Selected”的最顶端拿一张卡。产品负责人随时都可以修改“Backog”和“Selected”这两列，但不能动其他列。

“Dev”这列（被分成了两个子列）显示的是当前开发的工作，它的看板上限是 3。在网络术语中，看板上限对应的是“带宽”，生产周期对应的是“ping”（或响应时间）。

我们为什么非要把“Dev”这列分成“Ongoing”和“Done”两列？这是为了让产品团队知道哪些东西可以部署到产品环境了。

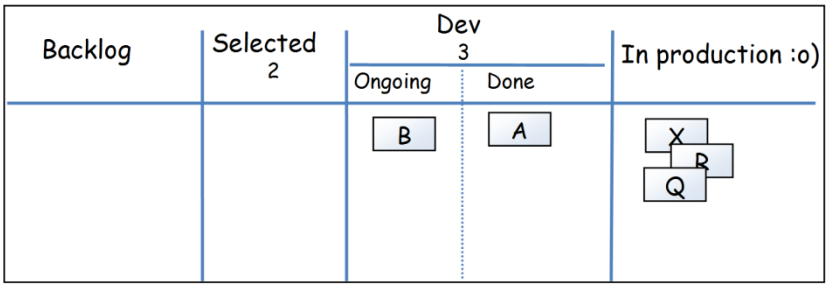
“Dev”的上限 3 是由两个子列共享的。为啥呢？举个例子吧，假设有两张卡在“Done”里面：



这意味着在“Ongoing”里面只能有一张卡。即便是团队有富余的人手*可以*做新任务，那他也不能做，因为看板已经到了上限。这会让他们有强烈的欲望，齐心协力把东西部署好，清空“Done”这一列，让流动更顺畅更迅速。 这种约束带来的效果是可以渐渐深入人心的 ——“Done”里面的东西越多，“Ongoing”能放的就越少——团队可以聚焦于做正确的事情。

单件流

单件流是“完美流”的一种场景。在单件流中，一个任务从板上从头流到尾，中间没有任何阻碍。这就是说在任何一个时刻，都有人为这张卡工作。单件流图大概是这样的；

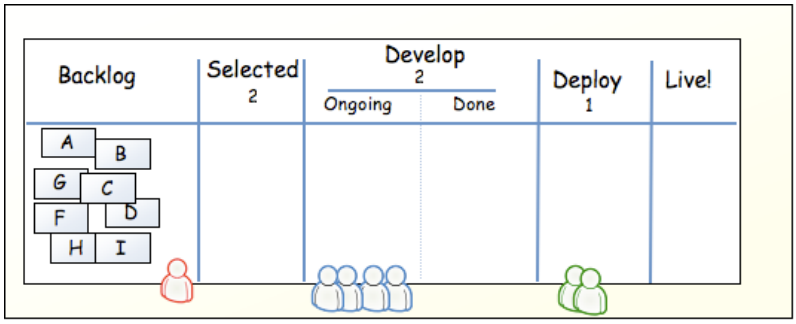


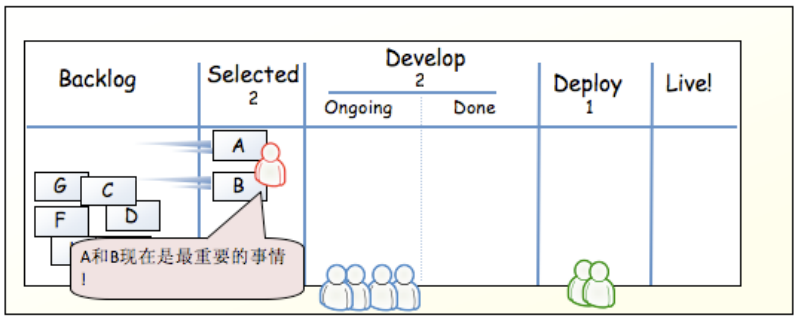
B 正在开发，A 即将部署。什么时候有空做下一张卡了，团队就去问产品负责人要最重要的事情做，他们会立马得到答复。如果这个理想的场景一直存在下去，我们就可以把“Backlog”和“Selected”这两列干掉，让生产周期*真正*短下来！

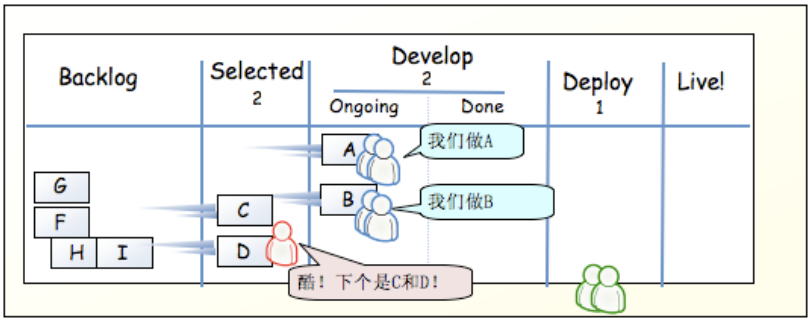
Cory Ladas 对此做了精彩的总结：“理想中的工作安排，是应该总是为开发团队准备好下一步要做的事情，既不能多也不能少。”

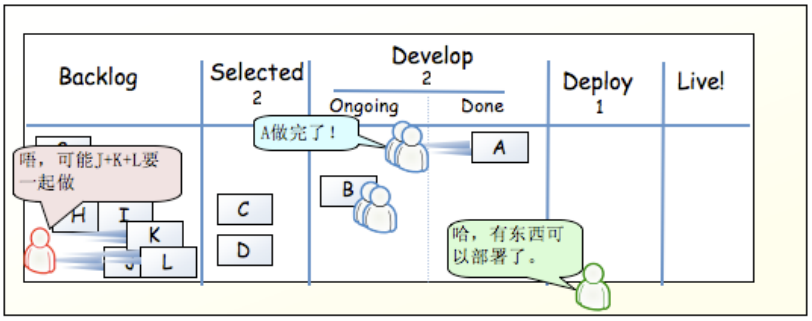
这里的 WIP 上限只是用来防止问题脱离控制，如果一切都在顺畅流动的话，WIP 上限也就用不着了。

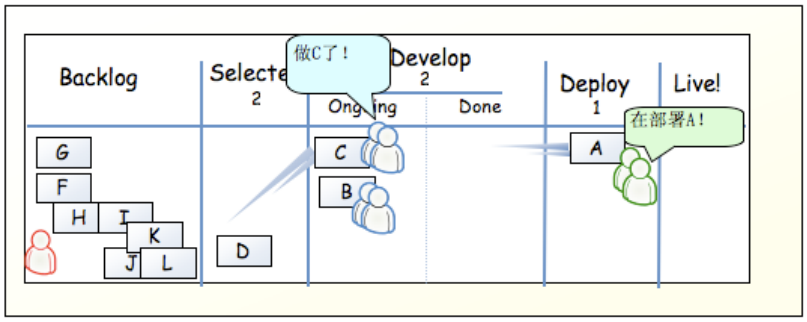
看板大陆上的一天

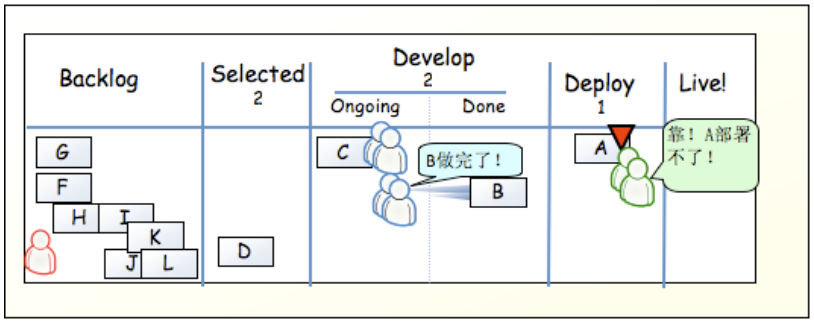

Backlog
Selected 2
Develop 2
Ongoing
Done
Deploy 1
Live!
A
B
G
C
F
D
H
I

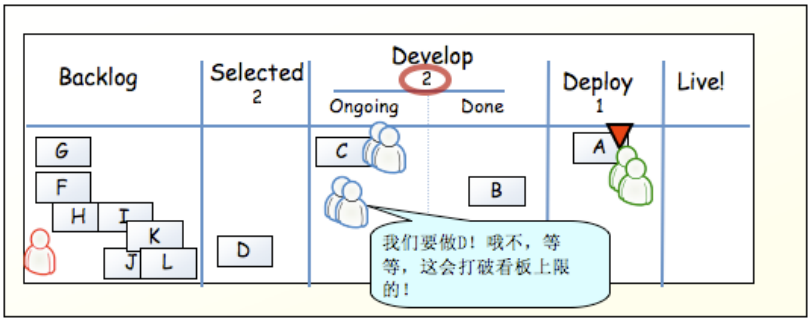

Backlog
Selected 2
Develop 2
Ongoing
Done
Deploy 1
Live!
A
B
G
C
F
D
A和B现在是最重要的事情！

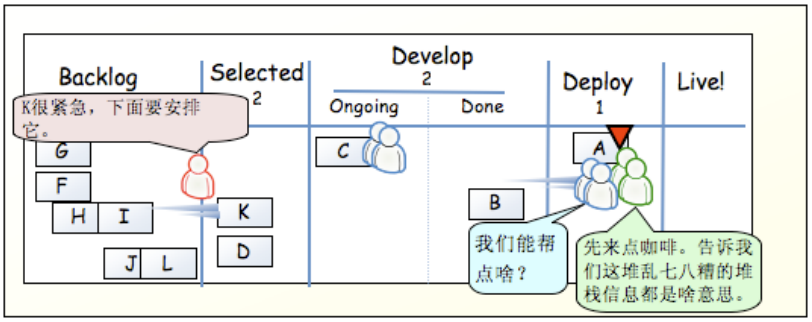

Backlog
Selected 2
Develop 2
Ongoing
Done
Deploy 1
Live!
我们做A
我们做B
G
F
H
I
C
D
酷！下个是C和D！

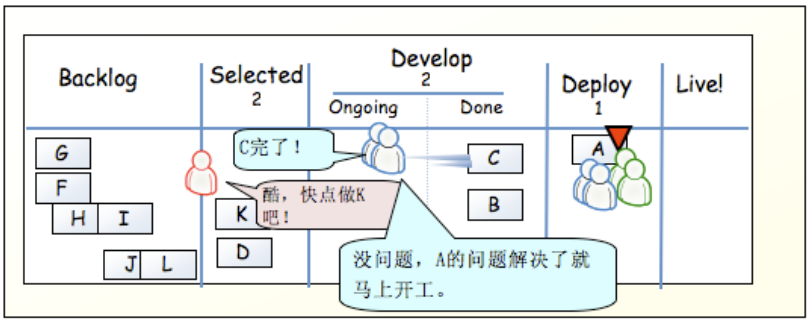

Backlog
Selected 2
Develop 2
Ongoing
Done
Deploy 1
Live!
A做完了！
A
B
C
D
嗯，可能J+K+L要一起做
H
I
K
J
L
哈，有东西可以部署了。

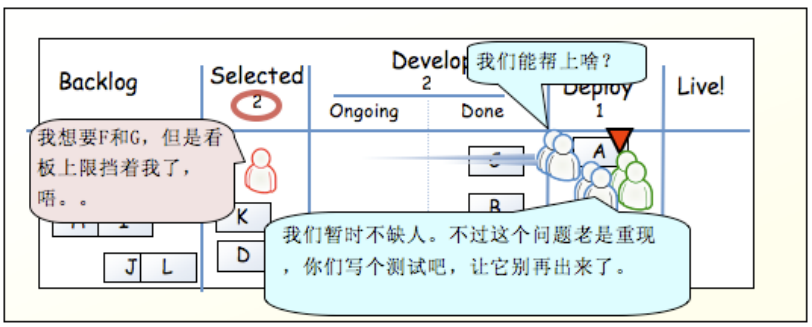

Backlog
Selecte 2
做C了！
Develop 2
Ongoing
Done
Deploy 1
Live!
在部署A！
G
F
H
I
K
J
L
C
B
D
A

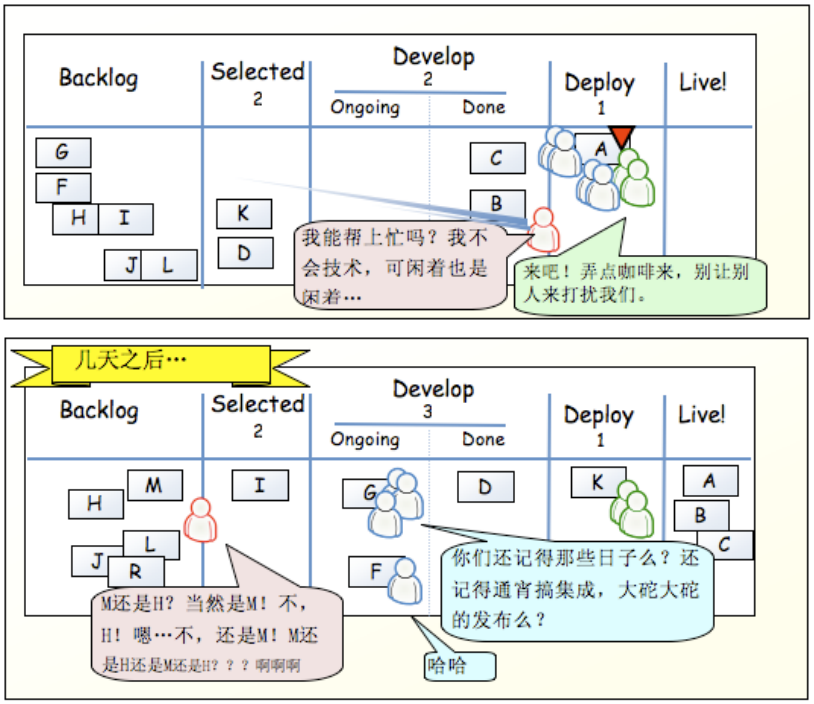

Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
G
F
H
I
K
J
L
D
C
B做完了！
B
A
靠！A部署不了！


Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
G
F
H
I
K
J
L
D
C
B
A
我们要做D！哦不，等等，这会打破看板上限的！


Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
K很紧急，下面要安排它。
G
F
H
I
K
J
L
D
C
B
A
我们能帮点啥？
先来点咖啡。告诉我们这堆乱七八糟的堆栈信息都是啥意思。


Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
C完了！
G
F
H
I
K
J
L
D
C
B
A
酷，快点做K吧！
没问题，A的问题解决了就马上开工。


Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
我们能帮上啥？
我想要F和G，但是看板上限挡着我了，唔。。
K
D
J
L
C
B
A
我们暂时不缺人。不过这个问题老是重现，你们写个测试吧，让它别再出来了。


Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
G
F
H
I
J
L
K
D
C
B
A
我能帮上忙吗？我不会技术，可闲着也是闲着…
来吧！弄点咖啡来，别让别人来打扰我们。
几天之后…
Backlog
Selected
2
Develop
3
Ongoing
Done
Deploy
1
Live!
M
H
L
J
R
I
G
F
D
K
A
B
C
M还是H？当然是M！不，H！嗯…不，还是M！M还是H还是M还是H？？？啊啊啊
你们还记得那些日子么？还记得通宵搞集成，大砣大砣的发布么？
哈哈

看板图非得看上去跟它一样么？

不，上面的图只是一个例子！

看板只规定了两件事，一个是工作流必须可见，另一个是 WIP 要有上限。它的目的是在系统中制造无障碍的流动，尽可能缩短生产周期。所以你要常常提出这类问题：

我们应该有哪几列？

每一列都代表一个状态，或是两个状态之间的（缓冲）队列。应该先从简单的开始，逼不得已的时候再增加。

看板上限应该是多少？

如果“你”的某一列已经到了看板上限，而你也没有任何事情可作，那就去找下游的瓶颈吧（也就是向右边找一下，看哪列里面有堆起来的卡片），找出来以后帮着把它干掉。如果找不到瓶颈，看板上限可能就太低了，因为设置上限的目的就是为了减少给下游瓶颈添乱的风险。

如果你发现有很多卡片安静地坐在某个地方没人动，也就意味着看板上限可能太高了。

- 太低的看板上限 => 发呆的人 => 低生产率
- 太高的看板上限 => 发呆的任务 => 长生产周期

看板上限有多严格？

有些团队把它当作军规（团队不能超过这个限额），有些团队把它当作指南，或是讨论触发器（上限是可以打破的，但对此应该有一个合理的解释，并就其发起讨论）。再重复一次，这是你自己的事。我不是告诉过你看板没有太多约束了么？

16

小结—— Scrum vs. 看板

相似性

- 都是既精益又敏捷。
- 都是拉动式计划。
- 都限制了 WIP。
- 都以透明的方式驱动过程改进。
- 都关注于尽早交付、频繁交付可发布的软件。
- 根基都是自组织型团队。
- 都需要把工作拆分。
- 发布计划都是根据经验数据（生产率/生产周期）不断优化的。

差异

Scrum	看板
规定了**固定时长的迭代。**	**固定时长的迭代是可选的。**计划、发布、过程改进等活动可以各有各的节奏。它可以由事件驱动，不用非要固定时长。
团队承诺当前迭代做完一定量的工作。	**承诺是可选的。**
用**生产率**作为计划和过程改进的默认度量手段。	用**生产周期**作为计划和过程改进的默认度量手段。
规定了**跨功能团队。**	跨功能团队是可选的。**可以有专职团队。**

任务必须分解 以便在 1 个 Sprint 里面能做完。	没规定任务规模。
规定了燃尽图。	没规定专门的图表形式。
间接限制（每个 Sprint 的）WIP。	**直接限制（每个工作流状态的）WIP。**
规定了估算。	**估算是可选的。**
不能往进行中的 Sprint 里面加任务。	**只要有人手富余就可以加任务。**
一个 Sprint Backlog 归一个团队所有。	一张看板图可以由多个团队或多人共用。
规定了三种角色（PO、SM、Team）	**没有规定任何角色。**
每个 Sprint 之间重置 **Scrum 板。**	看板图一直保留着。
规定了经过优先级排序的产品 backlog。	**优先级排序是可选的。**

好了，就到这里了。你现在已经知道区别了。

但是这本书还没完，精彩的部分才刚刚开始！把鞋子脱掉，跳到沙发上去找个舒服的姿势，跟 Mattias 一起畅游实践之旅吧。

围绕InfoQ中文站关注的领域设计话题

——线下定期举行的技术讨论沙龙

形式：1个主题，1+个嘉宾，N个参会人员。

参会人员：有决策权的中高端技术人员。

人数：限制人数，保证每个人的发言权利。

频率：每月一次。

城市：以北京为主，现已在北京、杭州、深圳、西安举办过多场。

QClub是由InfoQ中文站定期组织的一个线下技术交流活动，目的是让中高端技术人员有一个相对自由的交流思想和交友的平台。每次QClub只关注一个主题，邀请国内外在该主题领域有研究的技术专家分享其经验，但QClub更注重讨论，因为它认为讨论才是真理，从讨论中才能激发参与者的智慧和火花。

InfoQ Enterprise Software Development Community QClub

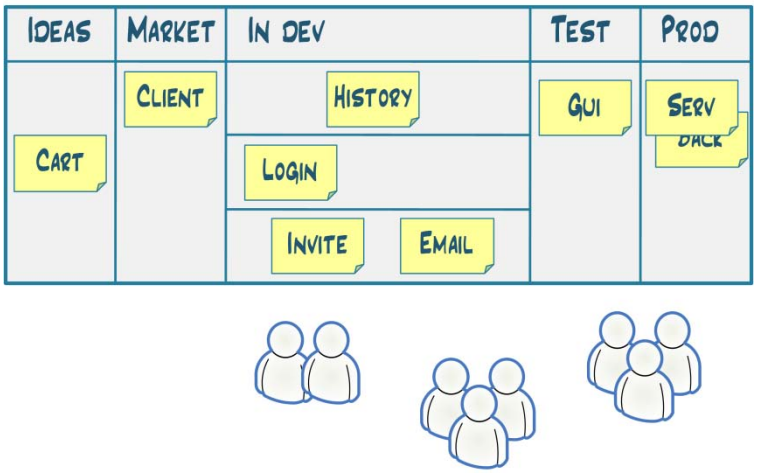
InfoQ官方微博：http://weibo.com/infoqchina

InfoQ豆瓣小组：http://www.douban.com/group/infoq/

讨论组：groups.google.com/group/infoqchina

编辑电邮：editors@cn.infoq.com

真实生活中的看板



接下来要讲述的是我们如何使用看板进行改善的故事。在刚开始的时候，到处都找不到有用的信息，Google 先生也一度让我们两手空空。而到了今天，看板已经成功的发展出了一套知识体系。我强烈推荐大家了解一下 David Anderson 的成果，例如“服务类别（classes of service）”。接下来是我们的第一个（也是最后一个）免责声明。不管你打算实施哪种解决方案，请确保它可以解决你的具体问题。我说完了。咱们继续吧。下面就是我们的故事。

/Mattias Skarin

17

技术支持的现状

倘若你曾经提供过 24*7 的电话支持 ,你就会全方位了解到管理产品环境所要担负的责任。不管问题是不是你引起的，你都得在深更半夜找出问题来源。没人知道问题在哪里，所以他们才给你打电话。你从来没有造过硬件，写过驱动，甚至客户用的软件也不是你写得 ,所以这更是麻烦事。你的选择往往就那么几种 :缩小问题范围，减少影响，保存现场，等到某人把问题重现后再解决。

响应能力和解决问题的能力是关键，当然还要快，还要不出错。

18

到底为什么要改变？

在 2008 年，我们的一位客户——一家北欧游戏开发组织——开展了一系列的过程改进活动。其中包括把 Scrum 推广到整个开发组织中，一点点消除阻碍开发团队交付软件的问题。等到软件运作良好、性能提升以后，改进的压力就落到了运维团队身上。先前他们基本上就是作壁上观，现在也越来越多地参与了进来，成了开发过程中的积极分子。

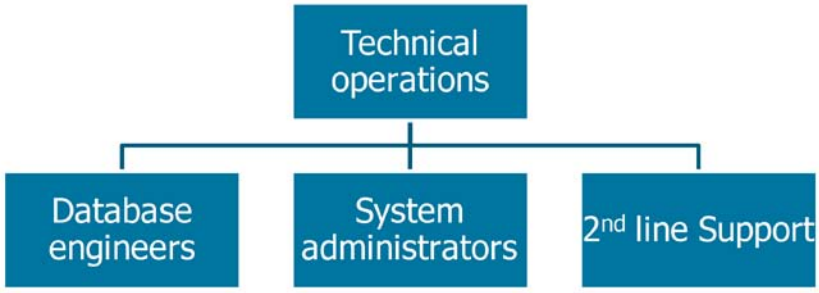


图 1. 运维团队包括三个小组：DBA、系统管理员、二线支持。

所以我们不能只帮助开发团队，否则技术支撑团队那边的核心运营设施改进工作就会拖了后腿。两手都要抓。

另外，改进工作在开发团队中取得进展以后，管理者就需要把更多的时间投入到创意的分析和反馈，这也意味着他们用来解决问题，及时划分任务优先级的时间就更少了。管理团队意识到，他们得在管理混乱到无可救药之前采取行动了。

19

我们从哪里开始？

因为开发团队算得上是运维团队的客户，所以一开始我们就先对开发团队做访谈。

从开发的视角看运维

我的问题是，“当你想到‘运维’这个词的时候，你首先会想到哪三件事？”下面列出了最常见的答案。

“知识全面”

“很擅长运营设施”

“他们想帮忙，但其实帮不上”

“项目时间太长了”

“他们的工作流系统太烂了”

“他们到底是干啥的？”

“一点小事就得来回发好几次邮件”

“找不到他们”

这就是开发人员眼中的运维。接下来对比一下，看看运维人员是怎么看开发的。

从运维的视角看开发

“你们为什么没用上平台现有的优势？”

“咱们把发布搞得轻松点行不行！”

“你们能把质量弄的好点不？我们都快不行了！”

“他们应该做出改变”——这是双方共持的一个论调。很明显，如果我们要引导大家解决共有的问题，这种思维方式是必须改变的。但是从积极的一面看，“很擅长运营设施”（暗示着信任对方的关键能力）这种说法也让我们坚信，如果我们创造出合适的

工作条件，“我们 vs 他们”这种想法是可以被修正的。消除加班，关注质量就是一个可行方案。

20

上路

我们要上路了，第一步该怎么迈呢？我们只有一件事情是很确定的，那就是“起点并非终点”。

我也是开发人员出身，所以我对运维这码事了解的也不多。我不想带来狂风暴雨一般的变化。我需要一个温和一点的方式，它本身要容易学，还能过滤掉无关的东西，让我们学到如何把事情做得更好。

候选方案有两个：

1. Scrum — 这个在开发团队中用得很好。
2. 看板 — 比较新，没检验过，但是跟精益原则很般配，而后者也正是组织中所欠缺的。

跟管理者讨论以后，我们发现看板和精益原则跟要解决的问题基本上是很吻合的。在他们看来，运维团队重排优先级的频率是以天为单位的，所以 sprints 的用途不大。于是看板就理所当然地成了出发点，即便它对于我们所有人而言都是新生事物。

21

团队启动

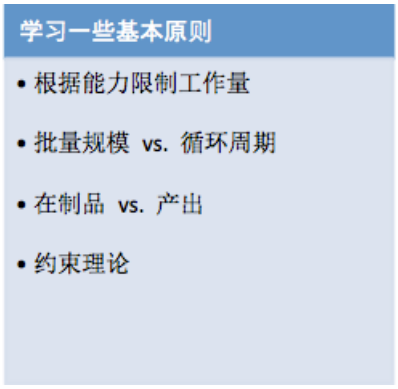
这两支团队该怎么启动呢？我们找不到现成的手册提供指导。如果这一步走错就糟透了。我们不但会错过改进的机会，而且也会跟这群一起为这个产品平台工作的、专业技能丰富的兄弟们疏远，这后果简直不堪设想。

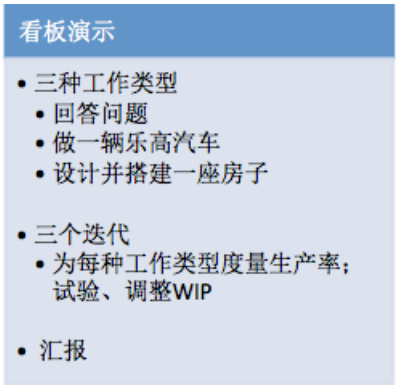
- 我们应该先开始再说么？走一步看一步？
- 还是先搞个研讨会？

答案很明显了——不就是搞个研讨会么？但怎么搞呢？把整个运维团队都拉过来参加研讨会的难度很大（有电话打进来的时候谁去接呢？）。最后我们决定办一个半天的研讨会，把内容精简一些，多做练习。

研讨会

研讨会有一个很大的好处，它可以尽早暴露我们的问题。它同时还为参与者提供了彻底放下防备心理的环境，有什么想法都可以直接拿出来讨论。并不是所有人都满怀激情地想要改变现有的工作方式——这点我们必须得承认。但大多数人还是比较开放，想尝试一下的。所以我们在研讨会上展示了一些最重要的精益原则，还模拟了一个小型看板。





会议结束的时候，我们用出拳的方式投票，看看大伙是不是愿意在工作中真实地体验一把。结果没人反对，于是我们就开始了。（译者注，“出拳”的原文是“fist of five”，即出拳头表示反对，出的手指越多表示越支持）。

很明显，实施看板的过程中，项目的相关干系人也会受到影响。固然这些变化可以带来好处——团队开始对无法完成的事情说“不”，开始坚持质量，把低优先级的东西从 backlog 中挪走。但无论如何，先沟通一下总是好的。

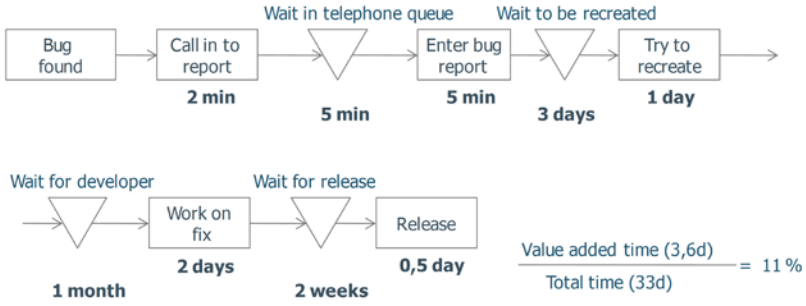
离一线支持最近的干系人是部门经理。他们已经参与了研讨会，所以对进一步落实看板还是很积极的。开发团队也一样（他们多少还是都期望改进的）。但到了运维团队那里，情况就不同了。他们最大的困难在于，每个人都已经被工作压得不堪重负，还要处理客户报上来的问题（公司已经承诺会解决所有问题）。如果我们要实施看板，强制推行 WIP 限额，这些困难就要想办法解决掉。

所以我们跟几个关键的干系人描述了一下愿景，实施后能获得的收益，还有可能发生的情况。就我看来，我们的想法还是很受认可的，有的领导强调说“如果我们最后能把这些问题都解决掉的话就太棒了”。

23

做出第一个图

用价值流分析可以有效帮助团队做出第一个看板图。价值流分析是一种价值链的可视化管理工具，它能够帮助使用者深入剖析工作状态、流动、循环时间。



不过我们的做法要简单得多：跟经理一起在纸上画了个看板图的样例。后来又检视了几遍，然后就开始用了。这个过程中通常要问这些问题：

- 我们有哪些类型的工作？
- 谁处理这些工作？
- 我们是否应该在各种工作类型之间共享职责？
- 在存在技能壁垒的情况下怎样共享职责？

因为不同类型的工作有不同的服务品质协议，所以还是让每个团队自行设计自己的看板图为好。于是他们自己做出了行和列。

还有一点也是比较关键的——到底要不要在各种工作之间共享职责。“我们是不是应该固定一部分人专门处理问题（应激性的工作），让其他人专注做项目（前瞻性的工作）？”我们打算先试一试职责共享。在做出这个决定的时候，有一个因素起了很大作用，那就是我们已经有了这样的认识：团队成员的自我组织和成长对于组织的可持续发展是不可或缺的。当然，这个方案也有些负面作用，它会让每个人的工作都受到干扰，但这已经是我们所能想到的最好方案了。另外提一句，我们搞研讨会的时候，来参加的团队就这个问题还真的自组织起来了。他们让一个人处理紧急请求，其他人处理耗时更长的事件。

第一个看板模型

下图是我们用的最基本的看板模型。这个模型跟大家常用的有点不一样，我们的想法是让卡片往上浮（就像水中的气泡一样），而不是从左往右移动。

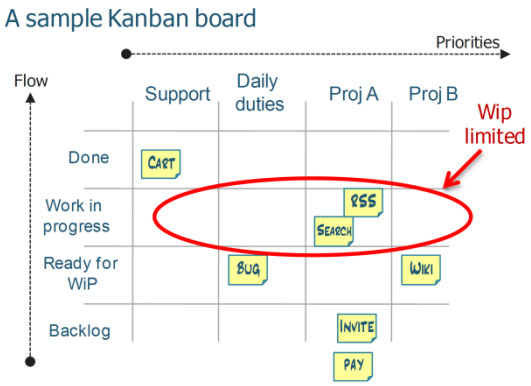


图 2.我们的第一个看板模型。优先级从左到右排，流动是从下到上。每一行的任务数相加就是在制品数量（如图中的红圈）。这个模型参考了 Linda Cook 的报告。

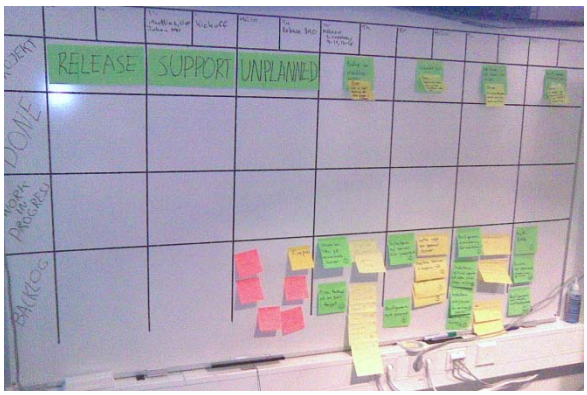


图 3. 系统管理团队用的第一个看板图

各行的含义

工作状态（行）	定义
Backlog	经理定下来的故事。
Ready for WIP	做过估算的故事。这些故事都拆成了不超过 8 小时的任务。
Work in progress	这一行有上限。我们一开始用的上限是 2*团队规模-1（-1 是因为大家要协作）。所以 4 人团队的 WIP 上限是 7。
Done	用户已经用上了。

各列的含义

工作类型（列）	定义
Release	帮助开发团队交付软件。
Support	其他团队提交过来的小块需求。
Unplanned	意料之外的工作，没有清晰的责任人。比如些许底层架构改进。
Project A	大型运维项目，例如更换试机环境的硬件。
Project B	另一个大型项目。

所有的看板图并非都是一个样子的。它们都是先从一个简单的骨架开始，持续演进。

24

设置第一个 WIP 上限

我们一开始设的 WIP 上限相当大，因为我们也没办法刚开头就猜得到该设成多少，所以就先把流动状态可视化出来，感受一下是否合适。后来每次找到合适的理由，我们都会调整 WIP 上限（其实只要看一眼看板图就清楚了）。

最开始给 WIP 设的上限是 2n-1(n=团队成员数量 ,-1 是为了鼓励协作)。为什么呢？一句话概括，就是我们也没更好的想法☺。另外每个人对此也毫无疑义。这个公式让团队有了既简单又合情合理的解释来应对那些要强加工作的人："...现在每个人都有两件事情要做，一个正在做，一个在等待，你怎么还能给他们再派活呢？"回想起来，其实在一开始的时候，随便设一个比较大的上限也都可以，在监控看板的过程中，还是很容易找出比较合适的数值的。

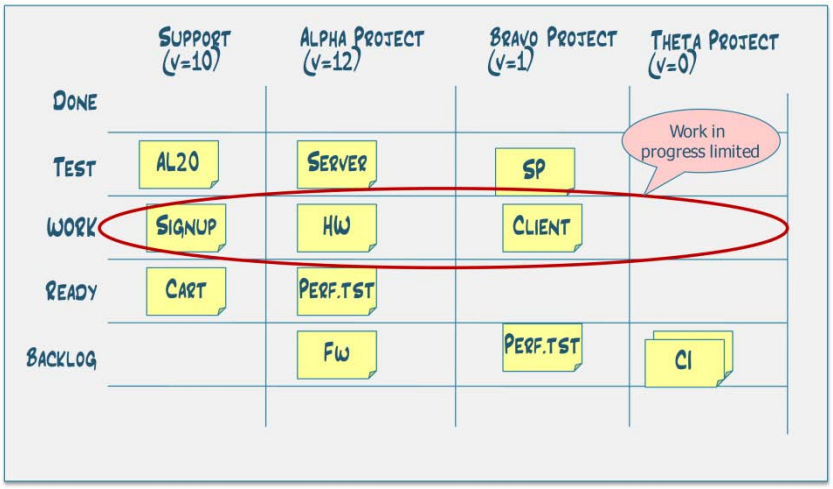


图 4.我们给 DBA 和系统管理团队设的 WIP 上限，每种工作类型各有一个上限。

我们还观察到一点，用故事点定义 WIP 上限是没意义的。这个太难跟踪了。最容易跟踪的单位就是卡片数量（并行开发的任务数）。

运维团队是每列定义一个 WIP 上限，因为在这个团队中，一旦并行的任务数逼近了上限，我们需要更快做出响应。

25

守住 WIP 上限

守住 WIP 上限听起来挺简单，但真要做起来就难多了。有些时候你就必须说“不”。我们试过多种方式来应对这种情况。

在看板图前面讨论

如果出现违背 WIP 上限的情况，我们就会把项目干系人带到看板图前面，问他们更想要哪些东西。刚开始的违规大多是因为大家缺少经验导致的，也有些时候是因为看待优先级的视角不同。举个比较典型的例子，具有特殊专业技能的人对于他本专业领域内的工作，就容易与别人看法不一样。我们只在优先级上出现过冲突，等把问题理清楚以后，大家到看板图前面讨论一番，冲突基本上就能解决了。

增加一个溢出区

有时候说“不”太冒犯别人，而且每张卡也都不是那么容易拿得下来的，于是当 WIP 到了上限以后，我们就把低优先级的卡放到“溢出”区里面。我们给“溢出”的任务定了两条规则：

1. 它们不能拿过去就算了；一旦有空我们还得把它们拿回来做完。
2. 我们扔卡的时候会通知大家。

才过了两周，我们就发现溢出的任务根本不需要再做了，在项目经理的帮助下，这些任务最终还是被拿掉了。

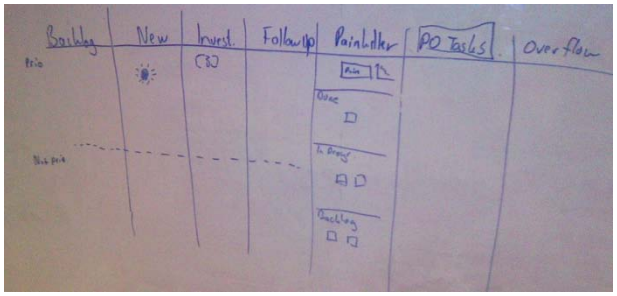


图 5.运维团队的看板图骨架；溢出区在最右端。

26

什么任务能放到看板图上？

我们早早地就定下规矩，不能把团队的所有事务都堆到板上。要是连打电话或者喝咖啡都得监控的话，看板就变成个噬魂怪了。我们是来解决问题的，不是来制造问题的☺。我们规定只能把超过 1 小时的任务跟踪起来，再小的任务就当作“白色噪音”了。这个 1 小时的限制还是相当有效的，我们只有少少几项措施从始至终没变过，这就是其中一个。（后来我们发现，背景噪音的影响也不可小觑，不过这是另外一个故事了。）

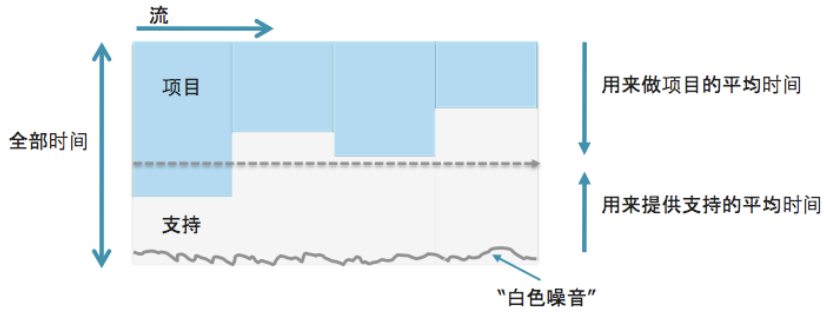


图 6.我们假定团队的全部时间全用来做两类工作：大的（做项目）和小的（提供支持）。跟踪项目生产率可以让我们对交付日期心里有数。我们假定“白色噪音”（小于 1 小时的任务、开会、喝咖啡、给人帮忙）一直存在。

27

怎样做估算？

这是个无休无止的话题，答案自然也五花八门：

- 定期做估算。
- 需要做估算的时候就做。
- 用理想天/故事点。
- 估算是不准确的，可以用 T 恤法（小、中、大）。
- 不要做估算，除非延期交付的成本大于估算的成本。

我们稍稍受了 Scrum 的影响（毕竟有过经验），决定先用故事点做估算。但在实际操作中，大伙还是把故事点等同于人-时了（这让他们感觉更自然一些）。一开始，大家老老实实地估算全部故事。后来经理们慢慢意识到，如果并发的项目数量很少，项目干系人就不用傻等着。他们还学会了重排优先级以应对突发事件。

结果就是项目交付日期不再成为大问题。于是经理们也就不再要求团队提前做估算——只有在担心会让别人干等着的时候才做。

记得当初有一天，有个电话打过来要求赶紧交活，经理压力很大，满口答应要“在这周末”交项目。那个项目也在看板图上展示着，进度很容易看出来（算一下完成的故事数），到周末顶多也就能做完 1/4。所以还得三周才行。看到这点以后，经理就把项目优先级改了一下，把其他正在做的任务全停了下来，为交付创造了条件。你一定要常常检查看板图啊☺

估算值代表什么呢？生产周期还是工作时间？

我们的故事点表示工作时间，也就是真正投入做故事需要多少小时，不是循环周期，不是日历上的时间，也不是等待时间。算好每周能“完成”的故事点数（即生产率），我们也能推断出来生产周期。

每个新故事我们只估一次，也不会在做开发的时候调整估算值，这可以尽量减少在估算上投入的时间。

看板给团队施加的约束极少，所以有很多种工作方式可供选择。既可以让团队按计划行事，又可以设置触发点，有了足够的任务再开工。

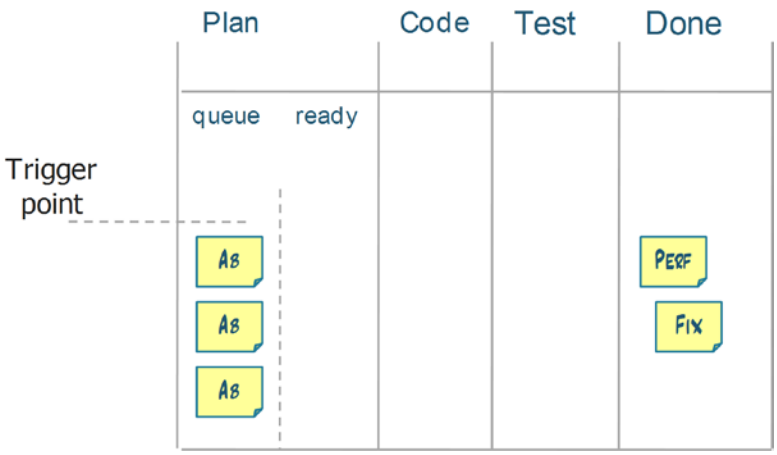
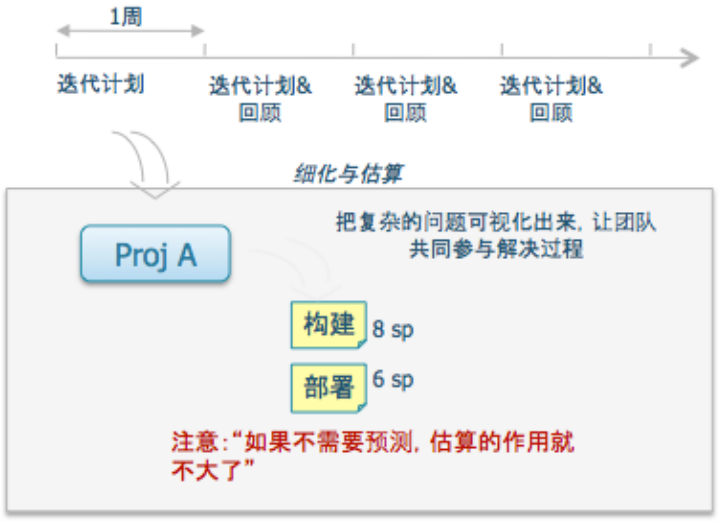


图 7.一旦 Backlog 里面满了三个任务，就会触发一个"计划/估算"事件。

我们还有两个周期性事件：

- 每日立会——大家都站到看板图前面来，说出问题，了解一下其他人的工作。
- 每周的迭代计划会议——目的是制定计划和持续改进。



这种做法对我们很管用。

每日立会

每日立会跟每日 Scrum 会议很像。所有团队（开发、测试、支撑）一起开完“Scrum of Scrums”会议以后再开立会。在 Scrum of Scrums 上，各支看板团队可以获得很重要的信息，比如哪些问题需要首先解决，哪个开发团队现在遭遇困境。刚开始经理们经常来参加，提一些方案，定一下优先级。后来团队慢慢地能够自我组织了，经理们来得也就越来越少了（但真有需要的时候可以随时叫来）。

迭代计划会议

我们每周开一次迭代计划会议，每次都在固定好的时间开，不然就总会被其他事情给占了。这个会还可以让我们多一些沟通。计划会议的日程一般是这样的：

- 更新图表和看板。（做完的项目就挪到“完成墙”上。）
- 回顾上周的工作。有哪些事情发生了？为什么会发生？如果怎样做会更好？
- 调整 WIP 限额（如果需要的话）。
- 任务分解，估算新项目（如果需要的话）。

这个迭代计划会议基本上就是把估算和持续改进合到一起了。因为有一线经理在，一些不太突出的问题当场就解决了，但有些比较复杂、涉及到基础设施的问题就得持续跟踪下去，很折腾人。为了解决这种恶心事，我们引入了一个方案：团队可以把两个“团队障碍”分配给经理们去解决。

规则如下：

1. 在任何时刻，经理都可以同时处理两个问题。
2. 如果配额满了，你可以把不重要的一个拿走，放进新的来。
3. 问题怎样才算解决由团队说了算。

这个方案效果不错。团队突然就看到了经理在帮助他们解决疑难问题。他们可以指着问题问，“现在搞定了没？”这些贴出来的东西不会石沉大海，也不会被其他突发事件冲掉。

我可以举一个例子。比如运维团队发现了一个 bug，他们需要定位是哪部份系统出的错，但是开发人员都忙着开发新东西，忙着冲刺，没法抽身帮忙，于是问题就堆积起来了。毫无疑问，这就让运维团队觉得开发人员对质量不够关心。

出现这种问题以后，团队首先上报到一线经理，然后再报到部门经理。部门经理就跟开发主管碰头，双方一致认为质量优先，商议得出一个轮换责任制方案——每个 Sprint 里面，都有一个开发团队随时待命，为运维服务。开发主管又去请示了他的领导，得到同意以后，列出一个联系人名单给了运维团队。运维的人对这种做法相当不信任，拿了名单就开始叫人，想试试好不好使。但开发主管早就对这些开发人员做好了心理工作，所以基本上是随叫随到。这就让运维团队解脱了。

29

哪种做计划的方法好呢

先讲一个故事

我还依稀记得有一次在一个团队里面发生的转机。那是在他们的第二次估算会议上。他们的那个项目让他们很纠结，不知道怎么做估算才好。未知的因素太多，会议一度无法继续。我没有参合进去，而是请他们自己调整估算过程，找出合适的方案。在经理的带领下，他们努力突破困难，设计了一种适合自己的估算方法。这件事情是个很重要的转折点，自此开始，他们真正融合成了一个充满自信的团队。而后他们便迅速地成长起来，我们不得不彻底放手，让他们自行发展。

过了两个月，他们做完一次回顾以后，经理过来找我，“我有个问题，”他指着看板图说，“我们现在几乎没有什么问题了，现在该做什么呢?”

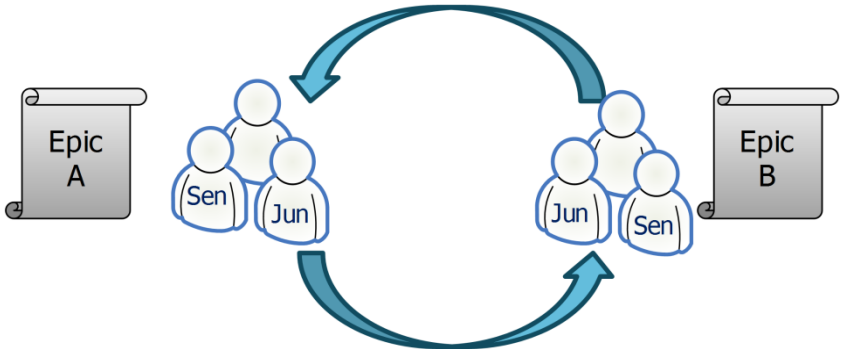
重塑计划会议

把全体成员凑到一起用计划扑克作估算，这种方式对任何运维团队几乎都不管用。如果你想要理由的话我就给你一些：

1. 团队内的知识极其不均衡。
2. 大多数时间只有一个人说话。
3. 大家都想解决当下的燃眉之急。

我所在的这些团队，通过不断的尝试和实践，独立创造出了两种不同的估算方法。每一种在他们那里都运作良好。

方法 1 – 轮换和评审



- 给每个项目或故事都分配两个人做估算，一个资深，一个资历尚浅（例如一个人很熟悉这个故事，另一个不熟悉）。这有助于传播知识。
- 其他人可以选择帮忙参与哪个故事的估算（每个故事最多 4 个人参加，保证讨论效率）。
- 每个估算团队都对手头的故事做任务分解，有必要的话也对任务做估算。
- 估算完以后就交换故事，检查对方的工作（每个团队留一个人解释估算的理由）。
- 搞定！

整个估算会议一般要花 45 分钟左右，从头到尾大伙都能保持精力充沛。交换故事以后，通常也就是稍稍调整一两处就行了。

方法 2 – 先让资深的人观其大略，再做估算

在估算之前，先让两个资深的人把要做的故事/项目大致上过一遍。他们在分析各种架构方案之后选定最适合的一个。做完这个以后，大家再凑过来，用架构方案当起始点，把故事拆分成任务。

图 8. 在迭代计划会议上，另一个团队的人评审任务分解的结果。

30

度量什么呢？

可以度量的东西很多 — 比如循环周期（从提出需求到交付的全部时间），生产率，队列，燃尽率……这里最关键的问题是，哪项度量数据可以**用来**改进流程。我的建议是先试验一下，找出最适合你自己的。我们在这方面有些教训，比如燃尽图对于 1～4 周的项目来说就太笨重了。即使没有燃尽图，在看板图上还是能看出基础进度信息的。（在 backlog 里有多少故事，做完了多少故事）

候选度量项	**优点**	**缺点**
循环周期	容易度量，不需要估算。始于客户，终于客户。	没有考虑规模。
整体生产率（统计所有类型的工作）	比较粗略，但是可以指示出改进和变化的方向。	对于某种特定类型的工作来说，无法从整体生产率上看出交付日期
每种工作类型对应的生产率	比整体生产率更为精确	只有从需求的起始点一直到交付点全程计算，这种数值才是有意义的。 比整体生产率要多花时间进行追踪。
队列长度	可以很快指示出需求的波动。容易可视化。	它不能告诉你波动是由不均衡的需求导致的，还是由不均衡的能力导致的。

		空队列可能实际上表示的是团队超负荷工作。

我们一开始度量的是“每种工作类型对应的生产率”和“队列长度”。前一项比较好算，也确实管用。后一项始终指示着我们的工作状态，抬头就看得到（只要你知道往哪看）。

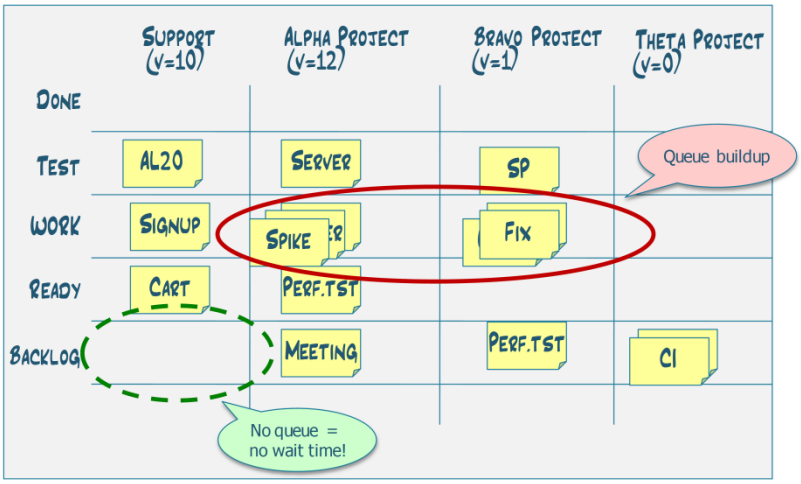
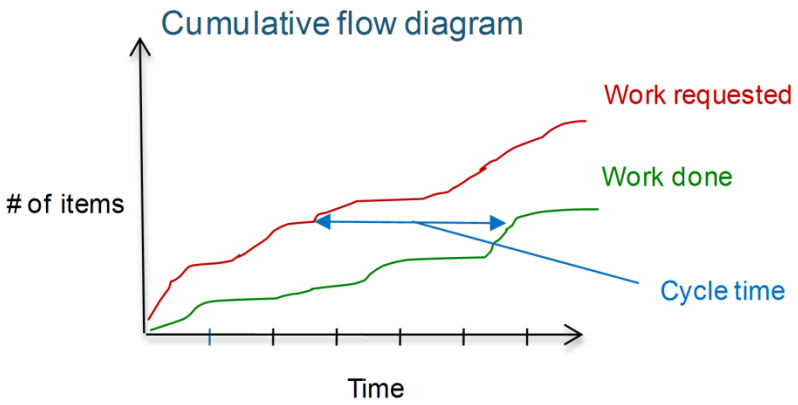


图 9.瓶颈和契机。红色区域显示出队列已满，测试成了瓶颈。而 support 那一栏的空队列表示一旦来了新的支撑工作就会马上被处理。它标识着团队的优秀服务品质。

我们没有用堆积流图，不过应该也很不错。



因为看板图和生产率图所提供的信息已经够用了，所以我们就没有用堆积流图——至少在当前成熟度不高的情况下还用不到。但我们依然可以很容易发现瓶颈、不均衡生产、超负荷，把这些问题解决掉，就已经够让我们在前六个月里忙活的了。

31

忽然一切都不一样了

在引入看板的三个月后，IT 部门中的系统管理员团队被授予了“最佳绩效奖”。同时，他们还在公司的总结表彰大会上位列“最佳贡献奖”三甲。这个总结表彰大会是在整个公司范围内举行的，每 6 周一次，这还是第一次有一个***团队***出现在前三名里面！可就在三个月前，这个团队还是 IT 运作的瓶颈，所有人都在抱怨它们。

很明显，他们的服务品质大大提升了。这是怎么做到的呢？

一切是从所有人齐心协力开始的。经理跟团队达成共识，有了明确的工作重心——经理保护团队不受无干事情的干扰，团队为质量和交付期限负责。这一切大概花了三到四个月才渐渐成形，但自那以后就顺流直下了。倒不是说所有问题都消失了（那我们就没活干了是吧？☺），我们现在面对的是新的挑战，“当团队已经不是瓶颈以后，怎么让他们保持持续改进的动力呢？”

促成自组织团队的措施中，有一点非常关键，那就是“每个开发团队配备一个运维的联系人”。这也就意味着，每个开发团队都在运维团队中有一个专属的联系人，给他们提供支持。正因为有看板，我们才能实现这种安排。看板让运维团队自我组织起来管理工作，避免了负担过重，推动了持续改进。在这之前，队列中的任务被谁领了说不定，领的人有没有能力做好也说不定，做完就算完了；这个过程中，一旦出现任何沟通上的问题，就得重新发起一个支持请求，一切从头再来。我们成功实施“一对一”概念以后，如果出现质量问题影响系统运行，支撑团队就可以迅速做出响应。

让我们开心的是，没过多久，这两支团队的沟通方式就进化了：运维人员开始跟他们熟悉的开发人员用 IM 交流，更擅长写字的人就发邮件，需要以最快方式解决问题的时候就打电话☺

在此之前

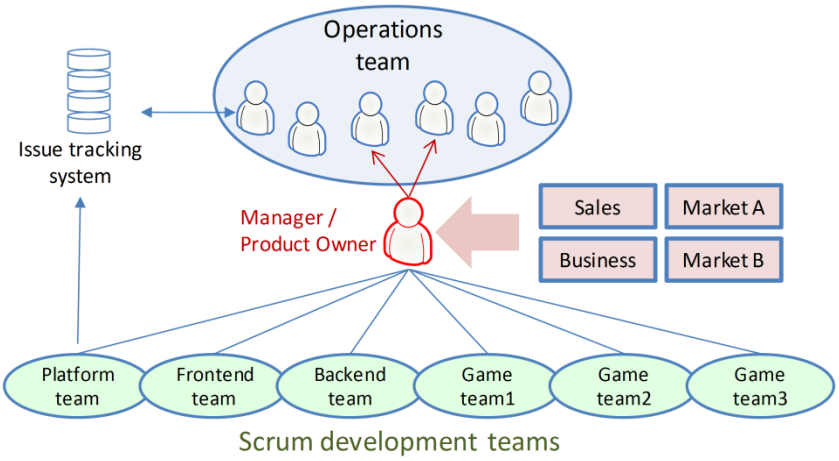


图 1. 在此之前：一线经理是主要的沟通接口。任何重要的事情都必须经过他才行。小问题——一般是开发人员的问题——会通过缺陷跟踪系统发送。很少有人与人之间的交流。

在此之后

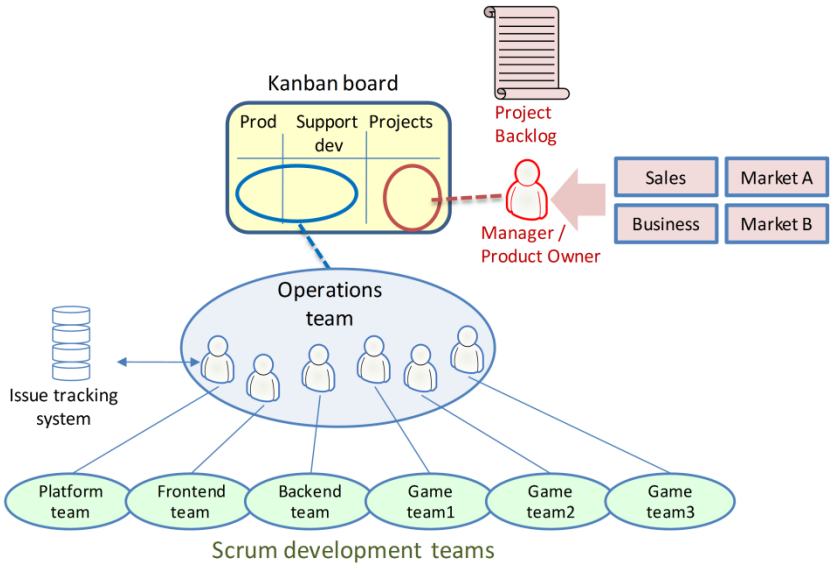


图 2. 在此之后："每个开发团队配备一个运维的联系人"方案落地。开发团队直接跟运维团队中指定的联系人沟通。运维团队用看板图自我管理工作。经理的关注点成了给规模更大的项目划分优先级，有困难的时候为团队提供支持。

这一切给团队的业绩带来了什么影响呢？

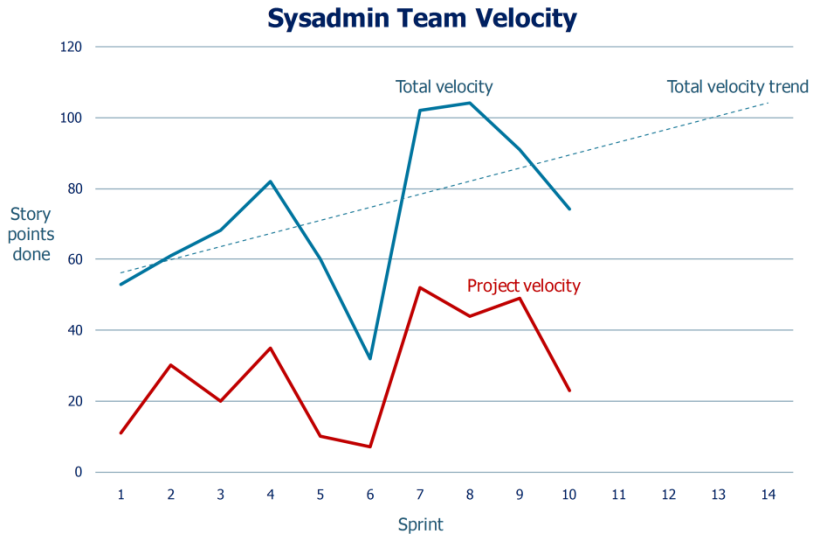


图 3. 整体生产率和项目生产率，以每周“完成”的故事点数度量。项目生产率表示的是为“projects”这一列做的工作（规模比较大的任务，比如升级硬件平台）。两次比较严重的生产率下降是因为 1)几乎所有人出去旅游了一周 ,2)有一次大型的发布。

所以，整体来看趋势还是比较乐观的。与此同时，团队还在知识分享上下了功夫，他们开始结对编程了。

我们再来看看 DBA 团队的绩效：

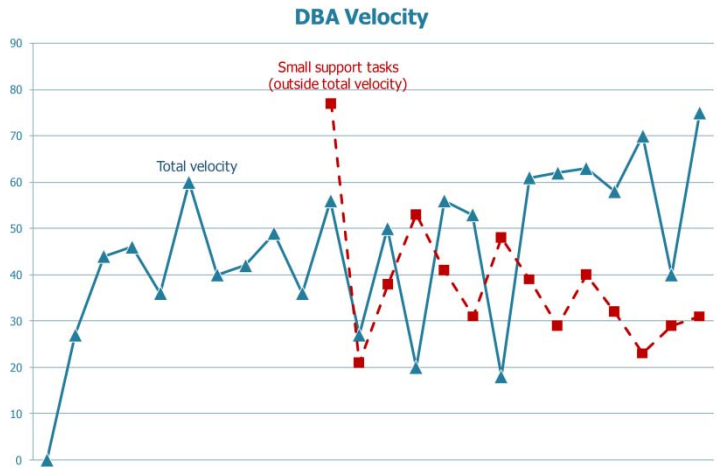


图 4 整体生产率和小型的支持任务。中间的下降是因为圣诞节。

生产率的整体趋势是上升的，但是也有很大的起伏。这些波动促使团队开始监控小型支撑任务的数量（这种任务通常耗时很短，不值得往看板图上放）。你可以看到，这幅图很明显地显示出了支持任务数量跟整体生产率的反比关系。

支持团队比其他两个团队（DBA 和系统管理员）实施看板的时间晚，所以我们到目前为止还没有什么可靠的数据。

成长历程

在刚开始的时候，随随便便就能找出要解决的问题来，但想发现从哪里下手改进可以获得最大的收益，这就很困难了。看板图提升了整体的透明度，不但让问题暴露得更明显，也促使我们思考更重要的问题，如流动、变数、排队。我们开始通过排队识别问题。实施看板四个月以后，经理们大幅削减了一直给团队带来伤害的变数。

团队从个体进化成自我组织的单元以后，经理们也意识到他们的领导力面临着新的挑战。他们要面对更多来自于人的问题——处理抱怨、定义共同愿景、解决冲突、谈判协商。这个过程始终与痛苦相伴——他们也承认，学习这些东西需要技巧和精力。但他们迎难而上，最后成为了更优秀的领导者。

32

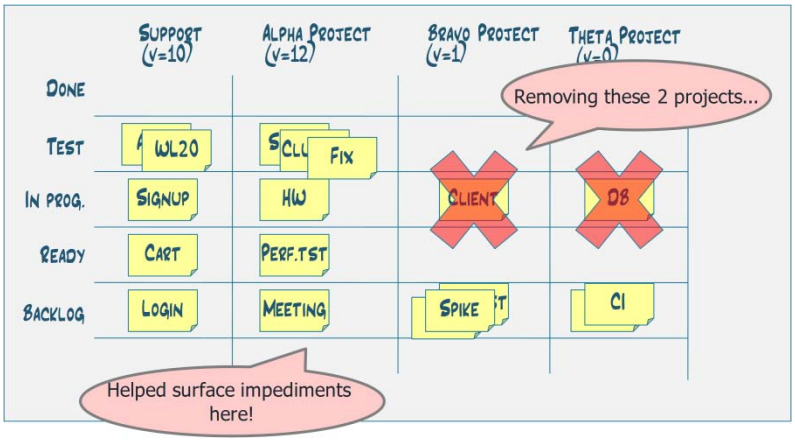
经验心得

随着在制品的减少，限制开始现形

所有团队一开始都给 WIP 的上限设了个比较高的值。那时候大伙的大部分精力都用来让工作流动起来，让组织能够得到必需的支持。

刚开头经理们还想着并行开展多个项目，没过几周，大家就看出来了，团队根本就没有富余的精力处理低优先级的项目。往看板上扫一眼就发现，低优先级的东西连碰都没人碰。经理就只好减少每个团队并发的项目数。

后来，流动顺畅起来以后，我们就开始严格控制 WIP 上限。我们把同时开展的项目（下图中的各列）数量从三减到二，再减到一。在这个过程中，团队之外的限制条件就暴露出来了。有些团队成员就报告说他们没有及时得到帮助，经理也开始把心思放到这类问题上面。



同时暴露的还有另外一个问题，其他团队引入的缺陷会严重影响本团队的效率。如果进入到队列里面的任务一直都有问题需要返工的话，要维持顺畅快速的流动就太难了。

在我们开始之前，这些问题一直隐藏着。这个事情的本质就是识别“哪个问题要优先解决”并达成共识。有了看板以后，每个人都能看到某个具体的问题如何影响了流动，就更容易积攒力量，解决跨越组织界限的问题。

图随时会变，别刻在墙上

所有的看板图都会发生变化。在团队最终找到最适合自己的那一款之前，通常都要修改上两三次。所以最开始就不要浪费大量时间来修理样式了。但一定要做得改起来方便。我们用黑色胶带来划分行列，这个很容易撕下来重贴，不管在墙上还是白板上都一样好用。我还见过用粗记号笔来画线的，这个可千万要确定能擦掉啊 ☺。

下面是优化布局的一个典型示例。因为刚开始的时候优先级会变化得很频繁，为了避免把一整列的贴纸前前后后地挪动，团队在每一列的列头上面都贴了张纸，表示优先级是多少。

图 14. 早期的看板形态，用贴纸指示优先级

不要害怕尝试和失败

在这次探险的旅程中，我有了一点感悟：探索是永无止境的。在某个地方失败过了，我们就知道这条路有问题了。人生本就是永无休止的探索和学习。没有失败就没有积累。我们犯过很多错（糟糕的看板设计、错误的估算、多余的燃尽图……）——但每次我们都能学到很重要的新知识。如果没有尝试，哪来的这些经验呢？

看板的成功实施也激励了管理团队和 Scrum 开发团队，他们也试着用上了看板。希望这本书对他们有帮助！

结束语

一切从回顾开始！

要思考的事情太多了？希望这本书可以驱散你眼前的迷雾。它至少对我们是管用的:o)

如果你打算改进流程的话，我们现在就可以帮你做个决定。要是你们没有经常做回顾，那就从回顾开始吧！你要确保回顾可以带来真正的变化。在需要的时候可以找个外援做主持。

一旦有了真正有效的回顾，你就迈上了通往持续改进之路的旅程，这条路的尽头是恰好适合你所在环境的流程——不管它是基于 Scrum，还是 XP、看板，还是几者的混合，等等。

不要停止尝试！

看板或 Scrum 不是我们的目标，持续学习才是。软件开发中最重要的事情就是快速反馈，这也是学习的关键点。要用好反馈！质疑一切、尝试、失败、学习、继续尝试。不要总想着一开始就能把事情做好，因为你做不到！选一个地方开始，不断改进就行了。

没有从失败中成长才是真正的失败。

但你还是能学到东西的，没有从“没有从失败中成长才是真正的失败”中成长才是真正的真正的失败……

祝你好运！

Henrik & Mattias 于斯德哥尔摩　2009-06-24

H: 我们写完了么？

M: 我觉得写完了，就到此为止吧。

H: 我们得告诉他们我们是谁吧？

M: 有道理。如果我们把自己描述的很有爱，可能还能有咨询的机会找上门呢。

H: 那就干吧！写完就完活了。

M: 是啊，不管咱们还是读者，都还有别的事情要干呢。

H: 嘿嘿，其实我马上就要休假了:o)

M: 你大爷的，别扯这事了行不。

作者简介

Henrik Kniberg 和 Mattias Skarin 是两位咨询师，在斯德哥尔摩的 Crisp 公司工作。他们喜欢从软件开发中技术和人的角度入手，帮助其他公司取得成功。他们已经让许多企业应用了精益和敏捷原则。

Henrik Kniberg

在过去的十年中，Henrik 曾任三家瑞典 IT 公司的 CTO，帮助过很多客户改进流程。他是一名 Scrum 认证讲师，经常跟精益和敏捷社区中的活跃人士合作，如 Jeff Sutherland、Mary Poppendieck、David Anderson。

Henrik 的前一本书《硝烟中的 Scrum 和 XP》拥有 15 万读者，是这类话题中最流行的书。他曾多次在国际会议中赢得最佳讲师奖。

Henrik 在东京长大，现在和妻子 Sophia 和三个孩子住在斯德哥尔摩。他在业余时间喜欢玩音乐、作曲，在当地乐队中做贝司手和键盘手。

henrik.kniberg<at>crisp.se

http://blog.crisp.se/henrikkniberg

http://www.crisp.se/henrik.kniberg

Mattias Skarin

Mattias 是一名精益教练，帮助软件公司从精益和敏捷的实施中获益。他的擅长领域涵盖了从开发到管理的各个层面。他曾帮助一家游戏开发公司把开发周期从 24 个月缩减到 4 个月，使公司重新建立了对整个开发部门的信任。他是看板的早期实践者之一。

他曾创建过两家公司。

Mattias 在质量管理专业拥有科学硕士学位，做了 10 年的核心业务系统开发。

他住在斯德哥尔摩，喜欢摇滚舞蹈、竞速滑雪。

mattias.skarin<at>crisp.se

http://blog.crisp.se/mattiasskarin

http://www.crisp.se/mattias.skarin

看板和 Scrum——相得益彰

翻译：李剑

审校：郭晓刚

美术编辑：胡伟红

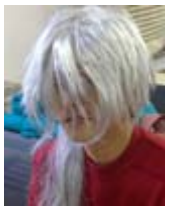
本迷你书主页为

http://www.infoq.com/cn/minibooks/kanban-scrum-minibook-cn

InfoQ 中文站新浪微博：

http://t.sina.com.cn/infoqchina

商务合作：sales@cn.infoq.com　13581658359

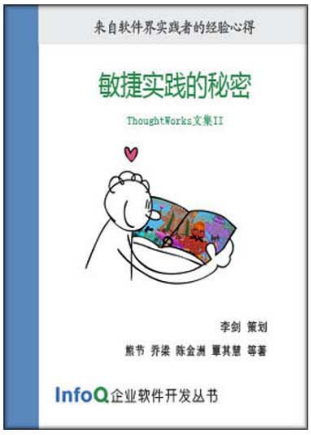


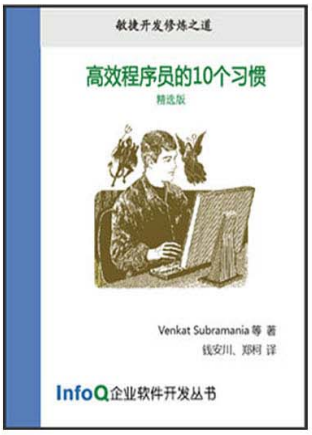
李剑，ThoughtWorks 高级咨询师，同时担任 InfoQ 敏捷社区编辑；在持续集成、测试驱动开发、重构等领域具有丰富的经验；多次为国内大型企业敏捷组织转型提供咨询和培训服务。他的译作包括：《硝烟中的 Scrum 和 XP》、《实现模式》、《精益和敏捷开发大型应用指南》等。

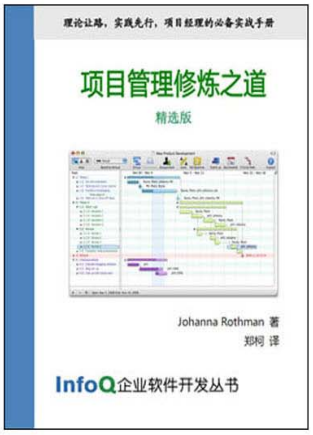
郭晓刚，一不小心就三张了，对编程这行算是开了点窍。编辑工作还做得挺开心，只不过戴上“编辑”的帽子阅读乐趣就少了。跟“企业开发”折腾了好些年，才发现这里面好玩的东西多了。千万不能早死，还没玩够呢。最近想在语义网和文档数据库上下功夫。

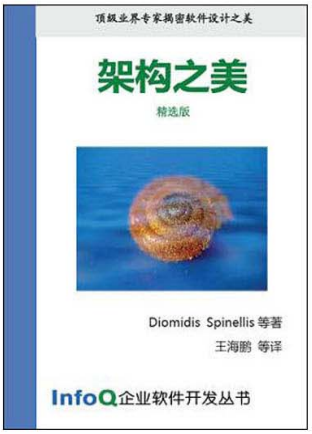
InfoQ企业软件开发丛书

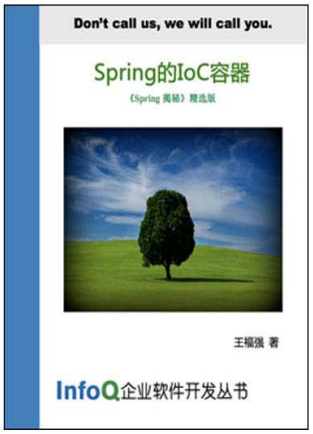
欢迎免费下载











商务合作：sales@cn.infoq.com　　读者反馈/内容提供：editors@cn.infoq.com