



看板方法 科技企业渐进变革成功之道

◎作者：David J. Anderson ◎译者：章显洲 ◎审校：路宁

译序

自有软件开发活动以来，在软件工程界和软件开发社区中，一直有不少先行者和实践者在积极探索如何更高效地组织团队进行高质量的软件开发活动。敏捷方法和精益方法正是近十几年来，从这波潮流中涌现的最精彩夺目的两项成果。敏捷方法和精益方法的大伞，覆盖多种软件开发方法学，其中最具代表性的有 Scrum、极限编程等，它们以《敏捷宣言》定义的价值观为基石，构成了这张大伞的一半幅面，另外一半幅面则由以精益价值观为基石的方法学组成，其中最具代表性的是由 Poppendieck 夫妇共同创立的精益软件开发方法（lean software development）。本书所介绍的看板方法，则是精益阵营中涌现的一种不可忽视的新方法。

看板方法，由 David Anderson 创立，它脱胎于大野耐一所创立的丰田生产方式（TPS），以及埃利亚胡·高德拉特（Eli Goldratt）的约束理论（TOC），并结合统计质量控制（SQC）、排队论（QT）、工业工程（IE）、软件成熟度模型（CMMI）等多个领域的知识，在软件开发社区中获得了极高的关注度，并迅速传播开来。

David 在从事软件开发管理的实践经历中，发现商业组织中的软件开发团队经常产生过载现象。这对软件开发者带来了深深的伤害，反过来也伤害了商业组织，因此他期望找到一种双赢的软件开发模式，寻找到一种系统性的途径，能够带来可持续的步调，既有利于软件从业者，也有利于商业组织。同时，作为变革推动者，David 还发现，在团队中导入新技术总是不可避免地会遭遇到阻力。他领悟到，必须寻找到一种能够使变革阻力最小化的方法才行，最好的策略便是从团队当前状况出发，采取逐步改善的变革引导方式。这两个方面，是 David 创立看板方法的两个基本动机。

看板方法采用了精益的思维范式，将软件开发视为一个价值流（Value Stream），并且基于拉模式来驱动其流动。“价值流”是精益思想和看板方法的基础隐喻，基于这个隐喻，引申出来的是一系列其他元素，例如，拉动（pull）、在制品（WIP）、批量规模（batch size）、前置时间（lead time）、阻塞（block）、瓶颈（bottleneck）、缓冲区（buffer）、吞吐量（throughput）、改善（kaizen）等。改善是精益和看板方法的精髓，它旨在通过持续性地实施系统性变更来优化生产系统。

看板方法的各种设计元素，为质量和过程中的问题提供了可见性，能够迅速暴露价值流中影响效能的问题，从而引导团队专注于解决问题以维持稳定的流动。通过帮助软件团队建立稳定的工作节奏，实现始终如一的可靠交付，看板方法能够在开发团队与客户、相关部门、供应商、价值流下游合作伙伴之间建立信任关系，从而建立具有高度协作、高度信任、高度授权和持续改进特征的组织文化。

软件开发中的看板方法本身还在不断发展过程中。作为本书的中文版译者，我非常荣幸能够把这本奠基之作介绍给中国的敏捷和精益软件开发社区。期待本书能够给社区带来新的视角和启迪，更期待国内社区对看板方法的实践能够反过来促进看板方法的进一步发展。

祝大家都能够精通保持平衡和持续改善的艺术！

章显洲
支付宝高级技术专家
2013年11月5日于杭州古荡

推荐序

看板在软件开发领域已获得越来越多的应用，而大卫·安德森是这个领域当之无愧的领导者，《看板方法：科技企业渐进变革成功之道》是他系统深入阐述看板方法的重要著作，是了解和实践看板的必备读物。

看板将软件开发带入到队列管理与流动的世界，它与其他敏捷方法的显著区别在于它是一种变革管理方法，在组织级渐进变革方面尤其具有优势。看板基于现状对现有方法做系统的渐进改良，阻力小，具有很强的暴露问题和提示改进机会的能力，能有效提高组织在完整价值流上的关注程度和绩效表现。在看板带来的渐进变革中可以结合其他敏捷方法，从而发展出更为适合企业自身特点的独特方法。

翻译是一项非常辛苦的工作，译者章显洲的辛勤工作加上他在看板方面的扎实实践确保了本书翻译的质量。我有幸审校本书的译稿，这对我来说更像是个学习的机会。本书非常容易阅读，读者可以从中找到许多实用的技巧。大卫对看板体系的搭建和使用极具洞察力，甚至一些不起眼的选择背后也饱含实践者的智慧，值得细细体会。如果你已经是敏捷的实践者，无论是在应用 Scrum 或是 XP，并在实践中遇到困惑，本书的不少内容会给你启发和指导。

我自认为在看板方面有颇多实践和思考，但参加大卫·安德森的培训，接触他本人，再认真审校《看板方法：科技企业渐进变革成功之道》译稿后我发现自己的远没有发挥出看板方法的威力。看板方法比大部分人想象的要系统和深入，这是一个值得深入学习的领域。审校本书对我来说是个转折点，我开始在已经应用看板的团队中对其做深入改造，并尝试在新团队和新项目中使用深入的看板方法。我利用看板建模过程分析团队信息，发掘真实流程。通过刨根问底式的沟通挖掘团队规则，借助工作项分类、服务分类以及队列的拉入条件将规则显示出来。我和团队一起识别看板系统的范围（可以为队列设置约束的那部分流程），针对看板系统建立推迟承诺的工作项拉入机制，并度量工作项在看板系统的前置时间，让每个人都看到自己的努力如何改善整个系统的表现。有了对现状的深入洞察，通过本书介绍的方法发现改进机会，我们开启了渐进式的良性过程。

团队追求前置时间的可预测性，这需要综合的努力和强大的问题解决能力，据此形成可信的服务承诺。我发现这会降低对时间盒迭代的依赖，避免追求迭代承诺时的一些副作用。团队追求看板系统的范围向下游扩展，按用户期望的频率部署，向上游扩展，利用看板的承诺方式改变用户与团队互动的方法，增进信任关系，利用推迟承诺为用户带来选择和灵活性。

我在组织层面开始尝试利用电子系统呈现和度量完整的价值流，价值流经团队所代表的服务。度量各个服务以前置时间反映的服务水平，评价通过前置时间分布反映的可预测性，以及以控制图反映的内部协作水平。组织内价值流的视角能够帮我们发现最值得改进的杠杆点，避免局部优化，也更容易把用户、团队、中层和高层管理者协调到同一个目标上来。

看板实践让我收获颇丰，也让我更加坚信看板方法将会给软件开发领域，特别是大型组织带来真正意义上的变革。相信读者在认真阅读、实践和思考后一定会获得与我类似的感受。

大卫·安德森对我说，经过最近几年的探索，他对看板方法有了进一步的理解，也对本书中的很多主题有了新的认识，现在的看板体系比以往更深入、更具体，这些从他最新的培训和演讲中便能看得出来。我很期待新版的《看板方法》能尽早面世。

路宁

2013 年 11 月于深圳

Agilean 联合创始人，咨询师，Lean-Kanban University 认证讲师

序

Foreword

我一直关注大卫·安德森的工作。第一次与他接触是在 2003 年 10 月，那次他送给我一本他的书《软件工程的敏捷管理：应用约束理论获得商业成功》¹。正如书名所示，这本书受到了艾利·高德拉特（Eli Goldratt）“约束理论（theory of constraints, TOC）”的深刻影响。后来，在 2005 年 3 月，我造访微软公司和他见面，那时他正在进行的研究工作给我留下了深刻印象，在其中他使用了累积流图（cumulative flow diagrams）的概念。再以后，2007 年 4 月，我有机会考察了他在 Corbis 公司实施的看板系统，深叹其突破性。

这里之所以罗列这些，是因为要说明一点，大卫的管理思想一直在不断发展，没有停歇。他不是停留在某个单一的理念上，然后试图迫使外部世界来适应这一理念。相反，他高度重视正在试图解决的问题的整体，对各种可能的不同解决方案保持开放的态度，通过实践不断地对它们进行检验，并反思其中的有效机制。在他的这本新书中，读者可以看见他经由这种方法所取得的成果。

显而易见，只有当方向正确了，速度才是最有用的。我相信大卫正在正确的方向上前进。这本关于看板系统的最新著作，令我特别激动。我在研究中也屡屡发现，较之 TOC，精益制造中的诸多理念对于产品开发更为直接有效。事实上，在 2003 年 10 月，我写信给大卫时就曾提及，“TOC 一个最大的不足之处，便是它忽略了‘批量规模（batch-size）’的重要性。如果第一优先级是找到和减少约束，那么一般而言，你正在解决的是一个错误的问题。”现在，我仍然相信这个判断是正确的。

2005 年我们见面时，我再次鼓励大卫超越 TOC 中“关注瓶颈”的思想。我向他解释说，“丰田生产方式（Toyota production system, TPS）”所取得的巨大成功，与发现和消除瓶颈无关。丰田是通过降低批量规模和变异性（variability），进而降低在制品库存（work-in-process inventory）来实现效能提升的。降低了库存，便可释放资金占用，带来经济效益，而正是看板这样的“在制品约束系统（WIP-constraining system）”，使其成为可能。

2007 年参观 Corbis 公司时，我看到了看板系统的一个具体实例，令人印象深刻。我向大卫指出，这已经远远超越了丰田式的看板方法。为什么这么说？虽然丰田生产系统经历过反复优化，几近完美，但它用于处理的是那些重复（repetitive）和可预测（predictable）的任务，这些任务的工期和延迟成本（delay cost）是类似的。在这种情况下，使用像“先进先出（first-in-first-out, FIFO）”这样的排程方法是正确的。当在制品达到限额时，在入口处阻挡新工作进入处理流也是正确的做法。但是，当要处理的工作非重复、不可预测、具有不同的延迟成本和任务工期时，这些方法就不是最优的了，这些情况恰恰是在产品开发中必须应对的。我们需要更先进的系统，本书正是在实践细节层面上对这些系统进行描述的第一本著作。

我想先简要提醒读者几点。

第一，如果你认为自己已经理解看板系统的工作方式，那么你理解的很可能就是和精益制造中使用的看板系统一样的那种方式。本书中的理念，远远超出了使用像静态的“在制品限制”、“先进先出调度（FIFO scheduling）”、单一的“服务分类（classes of service）”这样的简单系统。请仔细留意这些差别。

第二，不要认为这种方法只是一个可视化控制系统。使用一块看板（kanban board）²的方式，确实能够显著提升在制品的可见性，但这只是看板方法中很小的一个方面。仔细阅读本书，你会找到更多深层的内容。真正的洞见，寓含在诸如“到达和离开过程（arrival and departure processes）”的设计、“不可替代（non-fungible）资源”的管理，以及“服务分类”的使用等方面。千万不要受“可

¹ 译注：英文书名为《Agile Management for Software Engineering: Applying Theory of Constraints for Business Results》，国内由机械工业出版社于 2004 年出版中文版，是机械工业出版社的“软件工程技术丛书”之“前沿论题系列”中的一本，译者为韩柯等。

² 译注：此处的“看板”，真的是一块板（board），常用白板或墙面等来制作。本书中“Kanban/kanban”有多种意思，请读者详参第 2 章中的相关阐述。

视化”部分的干扰而错过了其中诸多微妙之处。

第三，不要因为这些方法看上去十分简易便生轻慢之心。看板方法的易用性，来自大卫对“什么能以最小的成本创建最大的效益”的深刻洞见。他已经敏锐地意识到实践者的需求，对方法中真正有效的部分给予了高度重视。最简单的方法最不容易遭到破坏，并且几乎总能产生最大的持续性收益（sustained benefits）。

这是一本令人振奋的重要著作，值得细细参读。读者从中的收获，将取决于阅读时的认真程度。对于这些先进理念，本书给出了最好的阐释。祝读者诸君如我一般深享书中的思想和乐趣！

唐·雷纳特森 (Don Reinertsen)

2010年2月7日于加州雷东多海滩 (Redondo Beach),
《产品开发流的原则 (The Principles of Product Development Flow)》作者

目录

Contents

第一部分 导论

第 1 章 解决敏捷管理者的困境	3
1.1 我对可持续步调的探索	4
1.2 我对成功变革管理的探索	5
1.3 从“鼓-缓冲-绳”转向“看板”	7
1.4 看板方法的出现	8
1.5 看板方法被社区采纳的过程	9
1.6 看板的价值是反直觉的	9
第 2 章 什么是看板方法	13
2.1 什么是看板系统？	15
2.2 把看板应用于软件开发中	15
2.3 为什么使用看板系统？	16
2.4 看板方法模型	17
2.5 识别看板方法的应用实施	17
2.6 作为权限授予者的看板	18

第二部分 看板方法的益处

第 3 章 一种成功秘诀	23
3.1 使用秘诀	24
3.2 成功秘诀和看板方法	35
第 4 章 在五个季度内，从最差变为最好	37
4.1 问题	38
4.2 可视化工作流程	38
4.3 影响效能的因素	39
4.4 明确过程策略	40
4.5 估算是一种浪费	41
4.6 限制在制品	41
4.7 建立输入节奏	42
4.8 达成新契约	43
4.9 实施变革	44
4.10 调整策略	44
4.11 寻求进一步的改善	45
4.12 成果	46
第 5 章 持续改进的文化	51
5.1 改善文化	51
5.2 看板方法会加速组织成熟度和能力的提升	52
5.3 社会学变革	57
5.4 文化变革也许是看板方法带来的最大好处	60

第 1 章

解决敏捷管理者的困境

Solving an Agile Manager's Dilemma

2002 年，我是摩托罗拉公司个人通信产品事业部（PCS）手机部门的一名开发经理，公司位于美国华盛顿州西雅图市的一个偏远小镇。我所在的部门原本属于一家创业公司，后被摩托罗拉公司收购。我们开发的是网络服务器软件，面向无线数据服务如空中（over-the-air）数据下载和空中设备管理服务等。这些网络服务器应用软件，和手机上的客户端应用紧密协同工作，是整个集成化系统的一部分；另外需要一起协同工作的，还包括电信运营商的网络以及后台基础设施中的其他组成要素，如计费系统等。公司管理人员无视工程复杂性、风险和项目的规模，为我们的项目完成时间定死了最后期限。我们写的代码是从原创业公司写的代码演变过来的，为了图快，开发人员走了许多捷径。一位资深开发人员坚持认为我们的产品只能算是一个“原型”。为了满足业务的需求，提高生产力和质量对于我们而言已迫在眉睫。

2002 年，我在日常工作中和写作前一本书时，一直在思考我所面临的两个主要挑战。第一，如何实现现在敏捷社区所称的可持续步调（sustainable pace），保护团队免受业务部门提出的无穷无尽的需求的困扰？第二，如何克服其中不可避免的变革阻力，将敏捷方法成功推广到整个企业内？

1.1 我对可持续步调的探索

My Search for Sustainable Pace

2002年，敏捷社区将可持续步调简单解释为“每周工作40小时”。《敏捷宣言的原则》告诉我们：“敏捷过程倡导可持续开发。出资人、开发者和用户应共同维持一个长期稳定的步调”。两年前，Sprint PCS公司的团队已经普遍认同了我的观点，“大型软件开发是一场马拉松，不是短跑冲刺”。如果要求项目成员在持续18个月之久的长期项目中保持步调，那么绝不能让他们在头一两个月就精疲力竭。必须为项目制订计划、预算成本、进行调度和估算，以确保团队成员每天合理的工作时间，避免他们过度疲劳。作为经理的我，所面临的挑战是，在实现这一目标的同时，还要能够确保满足所有的业务需求。

1991年，我首次在一家成立已经5年的创业公司里从事管理工作，该公司主要制造面向个人计算机与其他小型计算机的视频采集卡。我从CEO那里获得的反馈信息是，领导认为我“非常消极”。我们的开发能力已经达到极限，而领导却仍然要求增加更多的产品特性，这时我总是以“不行”断然拒绝他们。2002年以前，我形成了一种明显的模式：一直以“不行”这种拒绝的方式，抵挡来自业务方持续不断和善变的需求。

软件团队和IT部门似乎总会遭遇被其他组织任意摆布的悲惨命运。即使计划已经做得无懈可击、公正客观，这些组织仍然会对他们极尽软磨硬泡、恐吓威逼之能事。即使是基于深入分析、有多年历史数据支持的计划，在这些组织面前也会变得脆弱不堪。大多数团队则是既没有透彻的分析方法，又没有任何历史数据支持，在那些想要迫使他们承诺未知的（通常是完全不合理的）交付期限的权术手腕下，他们无疑势单力薄而只能任其摆布。

与此同时，软件工程师也已经基本上接受了疯狂的进度表和献身工作的荒谬言论，并对此司空见惯。软件工程师好像本不该拥有社交活动和家庭生活似的，这真是莫大的羞辱！我知道有太多人因献身工作而影响了和家庭成员的关系，并因此蒙受了不可弥补的损失。但要人们对那些身处主流的软件开发极客（geek）表示同情却很难。在我居住的美国华盛顿州，软件工程师的年收入仅次于牙医。就像20世纪20年代福特汽车装配线上的那些工人一样，其收入是美国平均工资的五倍，由于他们的报酬丰厚，没有人会考虑他们工作的单调性或身心的幸福感。很难想象，在诸如软件开发这样的知识工作领域会涌现出劳工组织；没人会从根本上去解决软件开发者日常遭遇的生理和心理方面疾病的问题。有些雇主很聪明地增加了额外的医疗保健福利，例如，按摩和心理治疗，偶尔还提供“心理健康”方面的病假，但他们不会去寻找这个问题的根本原因。一位供职于一家知名软件公司的技术作家，曾给出如是评论：“服用抗抑郁药并不涉耻辱，因为每个人都在这么做！”对于此种局面，软件工程师倾向于忍气吞声地接受命令，一边领取令人艳羡的高薪，一边默默承受身心之痛。

我想打破这种模式，想找到一种双赢（win-win）的方法，让我在说“好的”的同时，又能通过促进形成可持续步调来保护团队成员。我想回馈我的团队成员，让他们能够重返社交活动和家庭生活；通过改善环境，让那些开发人员不至于在年纪轻轻时，就因压力过大产生健康方面的问题。所以我坚定决心，尝试做些能改善这些问题的事情。

1.2 我对成功变革管理的探索

My Search for Successful Change Management

我思考的第二件事情是，在大型组织中引领变革这一挑战。在Sprint PCS公司以及后来的摩托罗拉公司，我均担任开发经理之职。在这两家公司中，确实都需要创建一种更为敏捷的工作方式。但

在这两家公司中，当我努力想将敏捷方法扩展到其他团队中时，却都感到颇为艰辛。

在这两个案例中，我并没有强势要求团队进行变革的权力。我都是应高级领导层的要求，试图去影响和推动变革，但我并没有任何行政职务上的权力。领导要求我在团队中对工作伙伴施加影响，就像我以前在自己的团队中所实施的那些变革一样。但是，那些在我的团队中明显奏效的方法，在其他团队中却遭遇抵制。存在这种阻力的原因，可能有很多方面，但最常见的原因是，其他每个团队的具体情况有所不同，在我的团队中使用的方法，必须根据其他团队特有的需求进行修改和裁剪。到 2002 年年中的时候，我得出了这样的结论：将某种软件开发过程强行施加在一个开发团队上，是无法奏效的。

一个过程需要根据每种具体情况进行适配调整。要做到这一点，需要每个团队拥有积极的领导者，而这点往往是团队所缺乏的。即使有了正确的领导者，如果没有框架指引裁剪过程来适应不同的环境，我依然怀疑能否发生什么重大的有益变革。如果领导者、教练或过程改进工程师缺乏这种引导，就很可能会基于自己盲目的信仰，主观地将变化强加给团队。这与因强行施加不适宜的流程模板而招致更大的愤怒和反对，是同样的道理。

在我写作《软件工程的敏捷管理》(Agile Management for Software Engineering)一书时，就已经开始着手解决这个问题。那时候我思考，“为什么敏捷开发方法能比传统方法产生更好的经济效益呢？”我试图用约束理论的框架对此进行研究。

在研究和写作那本书时我明白了，在某种程度上，每种情况都是独一无二的。怎么可能每个团队的每个项目会在同一时间同一地点遇到瓶颈呢？每个团队都是不同的：不同的技能要求、不同的团队能力和不同的经验。每个项目也都是不同的：不同的成本预算、进度、规模和风险状况。每一个组织也是不同的：不同的市场、不同的价值链环节。如图 1.1 所示，对我而言，这可能为研究变革阻力提供了一条线索。如果所倡导的工作实践和行为的变革，并没有带来能够被感知到的实际益处，人们就会抵制它。如果这些变化并没有改善团队成员感知到的约束或限制因素，那么他们一定会抵制这些变化。简而言之，脱离具体情境的变革建议，将会遭到身在其中、了解项目具体状况的工作者的拒绝和抵制。



图 1.1 为什么“一刀切”的开发方法学是无法奏效的

由此可知，由消除一个又一个瓶颈来不断演化发展出一个新过程，是最好的做法。这是艾利·高德拉特的约束理论 (Theory of Constraints) 的核心主题。尽管我认识到其中有很多东西可以学习，也感觉到约束理论的那些材料很有价值，但我还是按原定计划继续埋首于我的书稿写作之中。我十分清楚，那本书中并没有提供如何在更大规模上扩展实施这些理念的方法，因为关于变革管理的具体建议，书中几乎没有涉及。

本书第 16 章，将会解释艾利·高德拉特提出的方法。该方法的主旨是：识别并设法减少瓶颈因素，直到瓶颈因素不再对效能产生约束；消除一个瓶颈之后，又会涌现出另外一个新的瓶颈，如此循环不已。这种方法以迭代方式，通过识别和消除瓶颈，系统性地提升效能。

我认识到，可以将这项技术和精益方法中的一些理念合成（synthesize）在一起。首先以价值流（value stream）对软件开发生命周期的工作流程进行建模，然后建立一个可视化跟踪系统，此后，当新工作“流”经该系统时，通过跟踪其状态的变化，便可识别出瓶颈。在约束理论的基本模型中，具备识别瓶颈的能力是第一步。艾利·高德拉特已经针对“流”的问题（flow problems）开发出了约束理论的一种应用方法，并很别扭地将之称为鼓-缓冲-绳（drum-buffer-rope）。尽管名字有些别扭，但我意识到，也可将一种简化的鼓-缓冲-绳方案应用于软件开发中。

鼓-缓冲-绳一般被看成是拉动系统（pull system）这类解决方案的一个具体实例。正如将在第2章中看到的，看板系统是拉动系统的另外一个实例。拉动系统的一个有趣副效应是，它们通过将在制品限制在一定的数量范围内，来防止工作者的工作量过载。此外，只有瓶颈区域的工作者会维持满负荷，其他人则应该会有富余时间（slack time）。我意识到，拉动系统也许可以解决我面临的两个挑战。拉动系统可以让我逐步实施过程变革，（我希望）它能够显著减小阻力，同时有利于形成可持续步调。我决定尽早建立一个“鼓-缓冲-绳”拉动系统。我想去尝试增量式的过程演化，看它是否既能形成可持续步调又能减小对变革的抵制。

2004年秋，我在微软公司工作的时候，机会来了。第4章的案例分析中将对此进行详细介绍。

1.3 从“鼓-缓冲-绳”转向“看板” From Drum-Buffer-Rope to Kanban

在微软公司实施的“鼓-缓冲-绳”解决方案行之有效。变革阻力很小，生产力提高了两倍多，前置时间（lead time）下降了90%，而可预测性则提高了98%。2005年秋，我在巴塞罗那的一次会议上汇报了数据结果；还有一次是在2006年的冬天。我的工作引起了唐纳德·赖纳特森的注意，他专程造访我位于华盛顿雷蒙德的办公室。他想说服我，实施一个完整看板系统的各项条件已经成熟。

看板（kan-ban）是一个日语词汇，英文字面意思是“信号卡”。在生产制造环境中，这种卡片作为一种信号，通知生产过程中的上游工序继续生产。在这种生产过程中，除非从下游工序中获得看板信号，否则每个工序中的工人不准进行额外生产。虽然我了解这种机制，但是不确定将之应用在知识工作领域，尤其是在软件工程中，是否有用或可行。我明白，看板系统可以促成可持续步调。但是，我未曾注意到它作为一种能促进增量式过程改进的方法已享有盛誉。我也没留意到丰田生产方式（Toyota Production System）的创建者之一大野耐一（Taiichi Ohno）曾说过：“丰田生产方式的两大支柱，是及时生产（just-in-time）和有人工介入的自动化（autonomation）。驱动这个体系运转的工具，便是看板。”换言之，看板是丰田公司所使用的改善（kaizen，持续改进）过程的基础。正是这种机制，使得丰田生产方式可以运行起来。据我近五年的经验，现在对此已经确信无疑。

幸运的是，唐纳德取得了令人信服的论据，表明我应该从“鼓-缓冲-绳”实施方法转向看板系统，因为有一个深奥的理由：相比“鼓-缓冲-绳”系统，看板系统可以更为优雅地从一个瓶颈区域中恢复中断。但是，能否理解这点，对读者能否读懂本书来说并不重要。

回顾我在微软公司实施的最后一种解决方案，我意识到，如果一开始将之作为看板系统来看待，其结果也将是相同的。两种不同的方法产生同样的结果，这令我感到非常惊讶。既然过程的最终结果是一样的，就没有必要将之专门看成是“鼓-缓冲-绳”的具体实施。

于是我倾向于使用“看板”这个术语，而不是“鼓-缓冲-绳”。看板已经应用于精益制造（或丰田生产方式）中。相比约束理论，看板的知识体系已经被广泛采纳和接受。虽然看板是一个日语词汇，相比“鼓-缓冲-绳”，其含义也不够明显，但是看板更容易发音，更容易解释，而且事实也证明它更便于传授和实施，因此最终确定使用“看板”这一术语。

1.4 看板方法的出现

Emergence of the Kanban Method

2006 年 9 月，我离开微软公司接管 Corbis 公司的软件工程部门。Corbis 公司位于西雅图市中心，是一家私人持股公司，主营业务是图像和知识产权。受到在微软公司实施改进所取得的结果的鼓舞，我决定在 Corbis 公司实施一个看板拉动系统。结果令人鼓舞，由此也产生了本书中所展示的大部分理念。正是这些理念的扩展，包括可视化工作流程、工作项类型、节奏、服务类别、专门的管理报告和运营回顾等，共同定义了看板方法（kanban method）。

本书其余部分将要描述的看板（Kanban）（大写字母“K”开头）方法，是一种渐进式的变革方法，其中采用了看板（小写字母“k”开头）拉动系统、可视化和其他工具，促成将精益思想引入技术开发和 IT 运营中。这是一个渐进的（evolutionary）和增量式的（incremental）过程。采用看板方法，你可以根据具体情况实施过程优化流程，将变革的阻力降至最低，同时让参与者在工作中能够维持可持续的步调。

1.5 看板方法被社区采纳的过程

Kanban's Community Adoption

2007 年 5 月，瑞克·嘉伯（Rick Garber）和我在芝加哥召开的“精益新产品开发大会”上，向 55 名听众展示了来自 Corbis 公司的早期成果。同年夏天，在华盛顿特区召开的 2007 年敏捷大会上，我组织了一次开放空间会谈（open-space session）来讨论看板系统，大约有 25 人参与。两天后，与会者阿罗·贝尔斯（Arlo Belshee）在一次闪电演讲（lightning talk）中分享了他的“Naked Planning”规划技术。显然已经有其他人在实施拉动系统了。后来，我们组建了一个雅虎讨论小组，小组成员很快就增长到 100 多人。至写作本书时，讨论组已经有 1000 多名成员。开放空间会谈的数位参与者承诺，他们会在自己的工作中尝试应用看板方法，且他们的团队基本上都使用过 Scrum。那些早期采纳者中，最值得称道的是卡尔·斯科特兰德（Karl Scotland）、亚伦·桑德斯（Aaron Sanders）和乔·阿诺德（Joe Arnold），他们都来自雅虎公司。很快地，他们便把看板方法带到了分处三大洲的十多个团队中。开放空间汇谈中另一位需要提及的参与者是平谷贤治（Kenji Hiranabe），此前他已在日本着手开发看板解决方案。不久之后，他为 InfoQ 就此主题写了两篇文章，受到了高度关注和重视。2007 年秋天，《敏捷项目管理》的作者和敏捷项目领导力网络（APLN）的创始人圣杰夫·奥古斯丁（Sanjiv Augustine）到西雅图访问 Corbis 公司时，称我们的看板系统是“近五年内见到的第一种新型敏捷方法”。

2008 年，在多伦多召开的 2008 年敏捷大会共计有六场在不同环境中应用看板解决方案的演讲。其中一位是约书亚·克里夫斯基（Joshua Kerievsky），他来自 Industrial Logic（一家极限编程的咨询和培训公司），在演讲中他展示了如何根据业务环境来以类似的理念对极限编程进行调整和改善。那年，敏捷联盟提名将戈登·帕斯克奖（Gordon Pask Award）颁发给阿罗·贝尔斯（Arlo Belshee）和平谷贤治，以肯定他们对敏捷社区所作出的贡献。他们俩对看板方法都作出了有目共睹的贡献。

1.6 看板的价值是反直觉的

The Value of Kanban is Counter-Intuitive

在许多方面，知识性工作是和重复性的生产活动对立的。软件开发和制造业之间的差异在很多方面都表现出很大的不同。制造业具有低变异性（variability）；而软件开发极其易变，它利用变异性，通过设计上的创新来产生价值。软件自然是“软”的，一般能够很容易地以低成本进行变化，而制造业则围绕着很难变化的“硬”件开展工作。

人们怀疑看板系统在软件开发和其他IT工作中的价值，这是十分自然的。在过去这些年中，整个社区已经深切体会到看板方法是一种反直觉的（counter-intuitive）方法。无人预见，在Corbis公司，看板方法能够对企业和改善跨职能协作产生很大的作用（参见第5章）。

我希望能够展示“看板确实有效！”我希望能够说服读者，通过采用看板的简单规则，可以提高生产力、可预测性和客户满意度，同时还可以缩短交货时间。与此同时，随着协作性工作的增加，将有助于在整个组织中建立更多更好的跨职能协作关系，从而带来组织文化上的变化。

总 结

- ❖ 看板系统来自一系列称为拉动系统的实践方法。
- ❖ 埃利亚胡·高德拉特的“鼓-缓冲-绳”作为约束理论的一种应用，是拉动系统的实施方法之一。
- ❖ 我之所以探索拉动系统，来自两方面的动机：一方面是寻找一种系统性的途径，使团队在工作中实现可持续的步调；另一方面是寻找一种方法，能够以最小的阻力引入过程变革。
- ❖ 看板是丰田生产方式及其改善方法的支撑机制，能够带来持续改进。
- ❖ 第一个虚拟的软件工程看板系统，是从 2004 年开始在微软公司实施的。
- ❖ 从早期的看板实施来看，结果令人鼓舞。看板确实可以实现可持续步调，通过渐进增量的方式，最大限度地降低变革阻力，并产生显著的经济效益。
- ❖ 作为一种变革技术的看板方法，自 2007 年 8 月在华盛顿特区召开的 2007 年敏捷大会展示之后，开始广受社区关注。
- ❖ 在本书中，看板（kanban）（小写字母“k”开头）指的是信号卡，看板系统（kanban system）（小写字母“k”开头）指的是使用（虚拟）信号卡实施的拉动系统。
- ❖ 看板（Kanban）（大写字母“K”开头）方法指的是自 2006—2008 年间在 Corbis 公司涌现的渐进增量式的过程改进方法学。这些年来，看板方法在更为广泛的精益软件开发社区中得到了持续深入的发展。

第 2 章

什么是看板方法

What Is the Kanban Method?

2005 年 4 月初的时候，我在日本东京度假，恰逢春天樱花盛开之季。为了欣赏美景，我打算和家人第二次游览位于东京市区的皇居东御苑公园。正是在这里，我受到启发——看板并不只供制造业使用。

2005 年 4 月 9 日，周六，我和家人走过位于竹桥 (Takebashi) 地铁站附近的护城河桥，经由北面入口进入公园。周六的早晨阳光明媚，许多东京市民正尽情享受公园的宁静与樱花之美。

所谓花见 (hanami)，指的是坐在樱花树下一边野餐一边欣赏落英缤纷。在日本，这是一种古老的传统，也是一种感悟生命的美丽、柔弱与短暂的机缘。樱花短暂的生命，也是我们自身生命的某种写照，在浩瀚的宇宙中，我们的生命亦显短暂、柔弱和美丽。

盛开的樱花，与东京市区灰色的建筑形成鲜明的对比。东京市区十分喧嚣，人群熙熙攘攘，声音嘈杂，而公园却宛如水泥森林中心的绿洲。当我和家人跨过护城河桥准备入园时，一位肩上挎着包的日本老人走到我们身边。他从包中掏出一把塑料卡片，给我们每人发了一张。当看见我胸前兜着 3 个月大的女儿时，他迟疑了下，想是否也要给她发一张卡片。最后老人认为我女儿也需要，于是递给了我两张卡片。老人没有说什么，而我的日语也有限，所以我们没有交谈。走进公园后，我们找了一块地方开始享用家庭野餐。



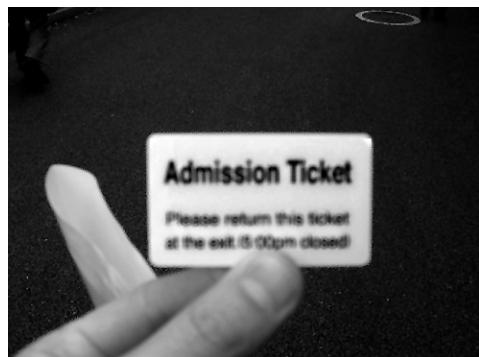
在阳光的照耀下我们度过了一个愉快的上午，两小时后，我们收拾起野餐的东西，朝位于大手町的东门口走去。近出口时，我们加入了一列在小亭子前排队通行的队伍中。随着队伍前移，我看到人们正在交还塑料入园卡。于是我在口袋里摸索了一会，找到了入园时发给我的塑料入园卡。走近亭子，我看里面有一位穿着整齐制服的日本女士。我们之间隔着一面玻璃，在台面位置上有一个半圆形的洞，类似于影院或游乐场的入场处。我把塑料入园卡通过柜台上的洞口递给了这位女士。

她手上戴着白手套，把我的塑料入园卡接过去后与其他入园卡一起放在了货架上并向我低头微笑致谢。没有收费，也没有解释为什么两小时前进来时要带塑料入园卡在身边。

这些入园卡是怎么回事？如果不是为了收费，何必发一张入园卡？我的第一直觉是，这肯定是一种安防方案（security scheme）。当傍晚公园闭园的时候，只需通过计算所有返回的入园卡，管理员就能够确知是否还有游客在园内逗留。转瞬间，我感觉如果真是这样，那这将是一个非常糟糕的安全系统。如果只给我一张而不是两张入园卡，似乎也可以。那么我那三个月大的女儿是作为行李看待还是作为游客看待呢？在这个系统中，似乎有太多变异性（variability）。有太多种出错的机会了！

如果这是一种安防方案，那么肯定起不到作用，每天都会误报有走失的游客。（顺便说一句，这样的系统不会轻易产生假阴性，因为那样还要生产额外的入园卡才行。这是看板系统很有用的一个基本属性。）与此同时，安防人员可能需要每晚出动，在灌木丛中搜寻走失的游客。不，这种做法肯定另有意图。随后，我明白了，皇居东御苑公园正在使用的是一个看板系统！

这次顿悟极大地启发我跳出制造业来看待看板系统。看起来，在各种管理场景中，看板令牌（kanban token）可能都有用处。



（感谢Thomas Blomseth拍摄了这些照片）

2.1 什么是看板系统？ What is a Kanban System?

看板（或卡片）的数量，等价于系统设置（核定）的流通能力。一张卡片与一个工作项关联。每张卡片都充当一种信号机制。只有获得一张自由卡片（free card）后，才可以开始新的工作项。这张卡片与该工作项关联在一起，跟随工作项在整个系统中一起流转。当自由卡片没有剩余时，就不能开始额外的工作。任何新到达的工作项必须在队列中等待，直到可以获得新的自由卡片。在某项工作完成后，和它关联在一起的卡片就与之分离而被回收。有了自由卡片，队列中的新工作项就又可以启动。

这种机制就是所谓的拉动系统（pull system），这是因为系统只有具备了处理的能力才能拉入新工作项，而不是基于需求将工作项推入系统中。由于流通中的信号卡数量表征了系统能力，所以只要恰当地设置能力阈值，拉动系统就不会出现过载（overloaded）现象。

在皇居东御苑公园案例中，公园本身就是一个拉动系统，游客便是在制品（work-in-progress），由流通中的入园卡数量限定系统能力。仅当有入园卡可供发放时，新到的游客才能入园。普通的日子，公园接待游客没有一点问题。然而，在繁忙的日子，如假日或樱花盛开季节的某个周六，公园就很受游客青睐。当所有的入园卡发放完时，新到的游客必须在园外的桥上排队等候，等待其他游客离园后回收的入园卡。看板系统提供一种简单、成本低廉和易于实施的方法，通过限制入园人数，来控制园内人数。这种技术可以让公园一直保持良好状态，避免因游客过多或拥挤对公园造成破坏。

2.2 把看板应用于软件开发中 Kanban Applied in Software Development

在软件开发中，使用虚拟看板系统来限制在制品数量。尽管“看板”的原意为信号卡（signal card），而且在大多数软件开发的看板实施中使用的也是卡片，但这些卡片实际上并没有作为一种信号起到拉入更多工作的作用。相反，它们仅起到代表工作项的作用。之所以称为虚拟的（virtual），是因为并没有使用物理的信号卡。拉入新工作的指示信号，是通过在制品限制指标（或系统能力上限）与以可视化方式表示的在制品数量之差推算出来的。有些实践者通过使用诸如便事贴或位于一块板上的物理插槽来实现物理看板。更多时候，信号是从工作跟踪软件系统中产生出来的。在第6章中，将描述一个把看板机制应用于IT工作的案例。

卡片墙已经成为敏捷软件开发中流行的可视化控制机制，如图2.1所示。无论是使用在软木公告板上钉索引卡片（index card）的方式，还是通过在白板上贴便事贴的方式来跟踪进行中的工作（WIP），都已经是司空见惯的事情了。值得留意的是，这些卡片墙本身并不是看板系统。它们仅仅是可视化控制系统（visual control system）。它们让团队以可视化的方式观察在制品并进行自组织（self-organize），无需项目经理或产品经理的指令，便可自行分派任务，将工作从待办项列表中移向完成状态。但是，如果其中并没有明确限制在制品数量，也不能在系统中发送信号拉动新工作项，那么这个系统并不能算是一个看板系统。第7章中会对此有更详细的介绍。

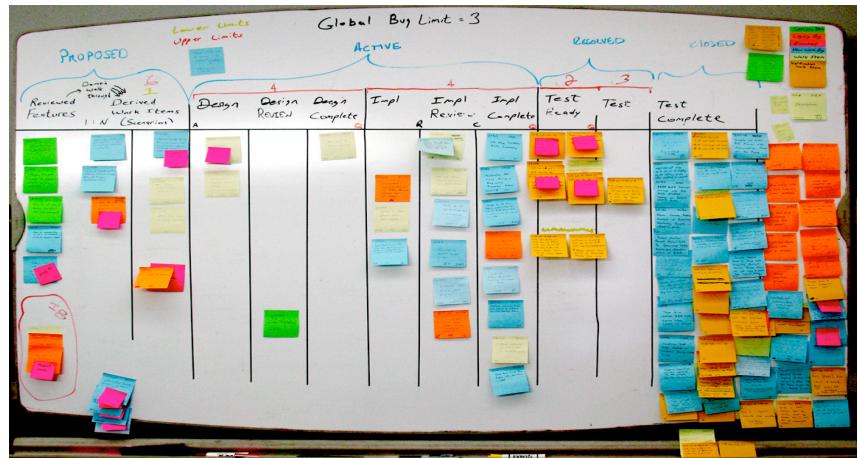


图 2.1 一个看板卡片墙

2.3 为什么使用看板系统？ Why Use a Kanban System?

后续章节将逐渐说明，通过使用看板系统，我们将团队的在制品工作项限制在一个设定的能力阈值内，根据已交付工作的交付速率来平衡提交给团队的工作需求（demand on the team）。这样做可以获得可持续开发的步调，让所有人都可以实现工作与生活之间的平衡。正如你将看到的，看板能迅速暴露那些影响效能的问题，因此，目前团队所面临的挑战是专注于解决问题以维持稳定的工作流。看板为质量和过程中出现的问题提供了可见性，使得缺陷、瓶颈、变异性以及经济成本等因素对流动与交付速率的影响变得更明显。单就使用看板来限制在制品工作项这一做法，就能促成更高的质量和更高的效能。流程改进和更好的质量结合在一起，有助于缩短前置时间（lead time），提高可预测性和准时交付能力。通过建立稳定的发布节奏，实现始终如一的可靠交付，看板能够帮助团队与客户、依赖的相关部门、供应商、价值流下游合作伙伴建立信任关系。

看板有助于实现组织文化的演进。通过暴露问题，引导组织聚焦于解决这些问题并消除其对未来的影响，看板能促进高度协作、高度信任、高度授权和持续改进的组织文化的形成。

事实表明，通过定期、可信赖、高质量地发布有价值的软件，看板能够提升客户满意度。事实还表明，看板能够提高生产率、质量，缩短前置时间。此外，有证据显示，在演进式文化变革中将涌现出更为敏捷的组织，看板在其中起到了关键的催化剂作用。

本书将致力于帮助读者了解如何在软件开发中使用看板系统，以及介绍如何和团队一起来实施这样的系统，以获得其带来的各项益处。

2.4 看板方法模型 A Model for the Kanban Method

为了能够在书中以文档方式记录看板方法（kanban method），我发现有必要正式地介绍模型。自看板出来至今，过去4年中虽然一直在不断演化，但我从来没有想过要进行这样的定义，直到今天。我担心它可能会因此变得僵化，也担心人们将之作为看板过程的真髓而教条式地引用，或者将之作为一种检验“我们在做看板吗”的测试方法，从而形成教条主义的思维模式。我希望，当试图定义看板方法的时候，也能够鼓励大家保持开放性思维，同时我们自己的认识仍将继续深化。我还会在 Limited WIP Society (<http://www.limitedwipsociety.org>) 站点上维护看板方法最新的定义。

2.5 识别看板方法的应用实施

Recognizing a Kanban Method Implementation

遵循看板方法的团队，会展现出 5 项核心特性。迄今为止，这 5 项特性已经在所有成功的实施实例中展现出来，包括我 2004 年在微软公司实施的“鼓-缓冲-绳”案例。

1. 可可视化工作流程。
2. 限制进行中的工作（work-in-progress）。
3. 度量和管理流动。
4. 明确过程策略。
5. 使用模型¹来识别改进机会。

看板方法还有 5 个附加的特性，我们期待能够在看板实施案例中发现它们。所有这些特性已在 Corbis 公司的实施案例中展现出来，过去 2 年内，在技术大会上曾展示过的和社区内有文档记录的大部分实施案例中，也已经展现出其中若干特性。

1. 根据延迟（机会）成本进行工作项的优先级排序。
2. 通过服务分类来优化价值。
3. 通过产能分配（capacity allocation）来管理风险。
4. 鼓励工艺和过程创新。
5. 定量化管理。

2.6 作为权限授予者的看板

Kanban as a Permission Giver

看板并不是一种软件开发生命周期的方法学，也不是一种项目管理的方法。实施看板时，需要当前已经有一些在运行的过程，这样便可应用看板来逐步改变当前运行的过程。

在敏捷软件开发社区中，对于这种提倡增量变化的渐进式方法一直存有争议。争议在于，看板方法建议开发团队不要采用一组预定义的方法或流程模板。但是，围绕两种流行的敏捷开发方法的一部分实践集合，相关的服务与工具行业已经发展起来。现在由于有了看板，个人和团队便具有开发自己独特的过程解决方案的权限，这便有可能削弱对这类服务的需求，产生对另外一套新工具的需求。事实上，看板也确实推动了一波新的工具厂商的兴起，它们迫切地想以一些新工具来代替现有的敏捷项目管理工具。相比较而言，新工具具备更强的可视化功能和可编程能力，可以随时根据特定的工作流进行调整配置。

¹ 看板方法中使用的通用模型包括约束理论、系统思考、对 W. 爱德华兹·戴明 (W. Edwards Deming) 在变异性 (variability) 方面的教导的解读，以及丰田生产方式中的浪费 (muda) 概念。在看板方法中使用的模型也处于不断发展的演化中，来自其他领域如社会学、心理学以及风险管理等方面的一些理念，在一些实施案例中也有所体现。

在敏捷软件开发方法发展初期，各种方法为何有效，社区的领导者们也常常不甚明了。大家提出了生态系统（ecosystems）的说法，并建议实施者遵循全部的实践，否则该解决方案有可能会失败。最近几年，出现了进一步鼓吹这种思维的负面倾向。一些公司已经发布了敏捷成熟度模型（agile maturity models），以设法评估实践的采纳状况。在 Scrum 社区有一种基于实践的测试，通常称之为诺基亚测试（Nokia test）。这些基于实践的评估方法的设计，旨在推动一致性和遵循度，而否认根据具体情境进行适应性变化的需要。看板方法则允许市场忽略这些基于实践的评估方案，它鼓励多样性（diversity）。

2007 年，有人到我在 Corbis 公司的办公室参观看板的具体实施情况。其中和敏捷软件开发社区关系密切的参观者都问及一个具有代表性的问题，“大卫，我们在这里转了一圈，已经看了 7 个看板，每个看板都不一样，每个团队遵循的是不同的过程，这么复杂，你怎么应付得过来？”对这种问题我一直是不屑回答的：“每个看板当然会不一样，每个团队的情况不同。他们必须演化过程，以适应自己的具体情境。”但我知道，这些过程均派生自同样的原则，由于团队成员都理解这些基本的原则，因此，即使将他们从一个团队转派到另外一个团队，他们也能够适应性地进行调整。

随着越来越多的人尝试看板，他们意识到，看板有助于解决在组织中推行变革管理时所遭遇的问题。看板可让团队、项目和组织展现出更好的敏捷性。我们认识到，在行业中，看板授予了一种权力，允许你创建根据具体的情境进行裁剪与适配的更优化的过程。看板授予了人们联系自身具体实际进行思考的权力。看板授予了人们保持与众不同的权力：你的团队可以不同于同一楼层的团队，



可以不同于隔壁楼层的团队，可以不同于旁边建筑里的团队，可以不同于隔壁公司里的团队。看板也授予了人们不死守教科书里那些条条框框的权力。看板方法最大的好处是，提供了一种工具，让我们可以解释（和辩护）为什么与众不同更好，为什么在那种情境下某种与众不同的选择才是正确的选择。受到谢帕德·费尔雷（Shepard Fairey）设计的奥巴马（Obama）竞选海报的灵感激发，为了强调这种选择策略，在为 Limited WIP Society 设计 T 恤衫的时候，我选用了丰田看板系统的创建者大野耐一（Taiichi Ohno）的头像作为图案。而设计“Yes We Kanban”的口号，是为了强调人们拥有这些权力。你有尝试看板的权力，你有修改自身过程的权力，你有保持与众不同的权力。你的具体环境是独特的，因此你应该根据自己的业务领域、价值流、需要管理的风险、团队技能和客户需求，进行剪裁、适配和优化，开发一种独特的过程定义。

总 结

- ❖ 任何情况下，都可以使用看板来限制系统内的某些事物的数量。
- ❖ 东京皇居东御苑公园使用了看板系统来控制公园内的游客数量。
- ❖ 可供流通的“看板”信号卡数量为在制品（进行中的工作项）设置了限额。
- ❖ 当前工作项或任务完成时，信号卡便被收回，用于将新的工作项拉入流程中。
- ❖ 在 IT 工作中，由于没有实际的物理卡片用于传递和定义在制品限额，所以我们（通常）使用虚拟的看板系统。
- ❖ 在敏捷软件开发中，常见的卡片墙并不是看板系统。
- ❖ 看板系统在工作场所中创建了一种积极张力，驱动大家去讨论问题。
- ❖ 看板方法（大写字母“K”开头的“Kanban”）利用看板系统（kanban system）作为改进的催化剂。
- ❖ 看板方法要求对过程中的规则进行明确的定义。
- ❖ 看板使用来自不同知识领域的工具，鼓励分析问题和探索解决方案。
- ❖ 通过不断探索发现影响过程效能的问题，看板方法可以实现增量式的过程改进。
- ❖ 可以通过 Limited WIP Society 的在线站点 <http://www.limitedwipsociety.org/> 获取关于看板方法的最新定义。
- ❖ 在软件开发行业，看板起到权限授予者（permission giver）的作用，鼓励团队根据具体情境制订过程解决方案，而不是教条式地遵循某种软件开发生命周期过程定义或模板。

第 3 章

一种成功秘诀

A Recipe for Success

过去 10 年间，我曾被要求回答这样的问题：“作为一名经理，当你接手一个现有的团队，尤其当这个团队并非以敏捷的方式进行工作，成员能力也参差不齐，甚至已经彻底陷入瘫痪时，应该采取什么样的行动？”通常，我被放在变革推动者（change agent）的管理职位上，因此，我只有面对挑战，发起积极的变革并迅速在两三个月内就取得成效。

作为大型组织中的管理者，我一直没能招聘属于自己的团队。我总是被要求接手已有的团队，以最小的人事变动启动一场变革，来提升组织效能。我认为，对于管理者而言，相比招聘一个全新的团队，这种情况更为常见。

在这个过程中，我逐渐形成了一种管理变革的方法。这种方法基于经验而来，其中包括从失败中汲取的诸多经验教训。这些失败教训，主要是指借助职权强制推行某些过程和工作流程。强制推行的管理方法往往失败。当我要求团队成员改变他们的行为，去使用某种敏捷方法如特征驱动开发（feature driven development）时，我遇到了阻力。我回应说，大家都无需害怕，因为我会提供必要的培训和辅导。然而，这样做，即使是最好的情况下，我也只能得到些默然的接受而已，而非取得真正深刻的制度化的变革成果。要求团队成员改变行为会令他们产生恐惧，降低他们的自信心，因为其中传达出的信息表明，现有技能的价值已经不再被看重。

我琢磨出了一种用来解决这些问题的方法，我将之称为成功秘诀（recipe for success）。成功秘诀中包含新任管理者对现有团队可以采取的若干行动指南。遵循这些秘诀，能够快速改善团队现状，而来自团队成员的阻力会很小。在这里，我要感谢唐纳德·赖纳特森（Donald Reinertsen）对我的直接影响，他贡献了秘诀里前面两步和最后一步，以及来自艾利·高德拉特（Eli Goldratt）的间接影响，他关于约束理论的相关著作和五步聚焦法（Five Focusing Steps）极大地影响了秘诀中的第四步和第五步。

秘诀中包含的六个步骤如下：

- 专注于质量；
- 减少进行中的工作（work-in-progress）；
- 频繁交付；
- 根据交付速率（throughput）来平衡需求（demand）请求量；
- 进行优先级排序；
- 消除变异性的根源，提升可预测性。

3.1 使用秘诀

Implementing the Recipe

秘诀中的各项内容，是按照技术职能经理能够依之操作的顺序排列的。“专注于质量”列在第一步，因为这是像开发经理或测试经理这样的管理者，或者其上司如拥有类似“工程总监”头衔的管理者，所能单方面控制和施加影响的。沿着列表向下，到“进行优先级排序”这一步，可控制性将逐步降低，而和其他上下游群体进行合作的要求则会逐步加强。优先级排序是业务部门的本职工作，而不是技术组织的工作，因此，不应该是技术经理职责范围内要考虑的事情。不幸的是，业务管理人员没能承担起这一责任，而把工作优先级排序扔给技术经理来做，然后反过来又责备技术经理做出糟糕的选择，这样的事情也相当常见。“消除变异性的根源，提升可预测性”之所以处在列表的末尾，是为了减少某些类型的变异性，必须进行行为改变。而要求人们改变行为是很困难的。因此，最好把消除变异性留在后面，等前面的步骤成功实施且组织氛围有所改变之后再行实施。如同第4章中将会看到的，为了促成那些先期步骤的成功实施，有时候需要先消除一些变异性的根源。其中的诀窍在于，要挑选那些几乎不需要行为改变而能为人们所欣然接受的变异性根源，须从它们开始入手。

“专注于质量”是最容易的一步，因为它是职能经理能够操控的一项技术性实践。其他步骤具有更大的挑战性，因为它们依赖于与其他团队彼此间的协议和合作。实施这些步骤，要求你具备口才、谈判、心理学、社会学和情商等在内的多项技能。就“根据交付速率来平衡需求请求量”达成共识，是至关重要的一步。而要解决团队成员之间在角色和职责方面的机能障碍问题，则需要更强的交际和谈判技能。因此，先寻找那些在你直接掌控之下，并且也知道解决它们对团队与业务效能将能产生积极影响的事情，这种做法是很有道理的。

能与其他团队建立相互的信任，可使很多艰难的改进成为可能。构建和提交缺陷很少的高质量代码，能够增进彼此的信任。通过有规律的构建活动来发布高质量的代码，也可以增加更多的信任。随着信任的增长，管理者便能收获得更多的政治资本（political capital）。这便能促进朝向秘诀的下一步前进。最终，大家都尊重你的团队成员，从而让你能够影响产品所有者（product owner）、市场营销团队、业务出资方（business sponsors）去改变他们的行为，大家一起协作，根据价值大小对开发工作进行优先级排序。

“消除变异性的根源，提升可预测性”是很难的一步。在一个团队变得更为成熟和效能水平提升到某种水平之前，都不应该进入这一步中。秘诀中的前四步，都能产生显著的影响。实施这些步骤，能够为一名新任经理带来成功。但是，要真正形成一种具有创新和持续改进的文化氛围，就必须在过程中不断消除变异性的根源。因此，秘诀中的最后一步是加分项。也正是这一步，将真正杰出的技术领导者与勉强胜任的普通管理者区别开来。

专注于质量 Focus on Quality

尽管“敏捷宣言的原则”（见参考文献 12）一文谈及了技艺（craftsmanship），其中隐含表明了对质量的关注，但是，敏捷宣言本身并没有提及质量。质量如果在宣言中都没出现过，为什么在成功秘诀中会处于第一位呢？简单来说，缺陷过多是软件开发中最大的浪费。这方面的数字相当惊人，有证据表明，其间可以有数个数量级的差异。卡珀斯·琼斯（Capers Jones）在 2000 年报道说，在网络泡沫期间，他调查北美软件团队的质量水平的结果显示，最多的每项功能有 6 个缺陷，最少的每 100 项功能少于 3 个缺陷，质量水平相差 200 倍。平均水平大约每 0.6~1 项功能有 1 个缺陷。这意味着，团队花费超过百分之九十的精力在修复缺陷上是一种普遍现象。有一个直接的证据，2007 年年底，看板方法的一位早期支持者亚伦·桑德斯（Aaron

Sanders) 曾在“Kanbandev”雅虎用户讨论组里报告说，他那时所在的团队，百分之九十的可用资源都花在了与缺陷修复相关的工作上。

鼓励提高初始质量 (initial quality)，会对高缺陷率团队的生产力和交付速率产生巨大影响。获得 2~4 倍的交付速率提升是很有可能的。如果团队真的很糟糕，那么通过“专注于质量”的做法，甚至都有可能获得 10 倍的交付速率提升。

软件质量有待提高是一个众所周知的问题。

不管是敏捷开发方法还是传统方法，对提高质量都有可取之处。应该综合使用它们。专业的测试人员应该做好测试。让测试人员来发现缺陷，防止缺陷遗留在代码中。要求开发人员编写单元测试代码，使单元测试代码自动化，以提供自动化的回归测试，这样也可以产生巨大的效果。看起来，要求开发人员先编写测试代码具有心理学上的好处。测试驱动开发 (TDD) 似乎确实能带来使测试覆盖更为完整的好处。但值得指出的是，我曾带领的纪律良好的团队，他们在功能编码之后再编写测试代码，质量也达到了业界领先的水平。然而，很明显，对于普通的团队，在功能编码前先编写测试代码，能够提高代码质量。

代码检查能够提高质量。无论是结对编程、同行评审、代码走查，或者完整的费根式检查 (Fagan inspections)，进行代码检查都是很有效的。代码检查能够帮助改善外部的代码质量和内部的代码质量。代码检查最好经常做，并且以小批量进行为好。我建议团队成员每天至少花 30 分钟进行代码检查。

协作式的分析和设计，能够提高质量。团队一起分析问题和设计解决方案，产出的质量会更高。我建议团队成员召开协作式的分析与设计建模会议。设计建模会议应该以小批量的方式每天进行。斯科特·安布勒 (Scott Ambler) 将此称为敏捷建模 (Agile modeling)。

使用设计模式 (design patterns) 能够提高质量。设计模式总结了对已知问题的已知解决方案。使用设计模式能够确保更早地获取更多的信息，使设计缺陷在软件生命周期的早期就得以消除。

使用现代开发工具也能够提高质量。许多现代开发工具都包括静态代码分析和动态代码分析的功能。对每个项目，应该把代码分析的开关打开，不断进行代码优化。这些分析工具，可以防止程序员犯低级错误，如安全漏洞这类众所周知的问题。

更为奇特的现代开发工具，如软件产品线 (software product lines) (或软件工厂) 和领域专用语言 (domain specific languages) 也能够减少缺陷。软件工厂可将设计模式封装为代码片段，从而降低在录入代码时引入缺陷的概率。它们还可以用于自动

化编码，取代重复性的编码任务，这样，在代码录入时引入缺陷的概率又进一步降低了。使用软件工厂还可以减少对代码检查的依赖，由于以软件工厂方式自动产生的代码其质量是已知的，因此并不需要对之重新进行检查。

这些建议中的后面几个确实已经属于整个秘诀实施过程中“降低变异性”的部分。使用软件工厂，甚至是仅使用设计模式，就是在要求开发者改变行为。而使用专业测试人员、测试先行、自动化回归测试以及进行代码检查，则可以获得更多的回报。

减少进行中的设计（design-in-progress）的数量，能够提升软件质量。

减少在制品并频繁交付 Reduce Work-in-Progress and Deliver Often

2004 年，我在摩托罗拉公司曾和两个团队一起共事。两个团队都为手机应用程序编写网络服务器端代码。一个团队的工作任务是开发服务器端软件，面向铃声、游戏和其他应用与数据的空中（over-the-air，OTA）下载。另外一个团队的工作任务是开发面向空中设备管理（OTA DM）的服务器端软件。两个团队都使用特性驱动开发（FDD）方法。两个团队的规模也大致相同，包含 8 名开发人员、1 名架构师、5 名测试人员，以及 1 名项目经理。他们都与市场人员共同工作，由团队自身负责分析和设计工作。此外，还有相应的团队为这两个项目团队提供用户体验设计和用户文档（技术写作）服务。

在制品、前置时间和缺陷

图 3.1 所示的是 OTA 下载团队项目工作的累积流图。累积流图是描绘处于某个给定状态的工作量的面积图（area graph）。这幅图中显示的状态包括：库存（Inventory），是指待办项或队列中那些尚未开始的需求项；已开始（Started），是指已经向开发人员解释的需求；已设计（Designed），是特指那些 UML 序列图已经绘制好的需求项；编码完成（Coded），是指那些已经实现了序列图上的方法的需求项；完成（Complete），是指需求项的所有单元测试已经通过，代码也已经进行了同行评审，并且团队主开发人员也已经认可编码，并且确认其可进入测试。

图 3.1 的第一条曲线，显示的是项目范围内的需求特性数量。需求特性分两个批次由业务方送来。第二条曲线显示的是已开始（Started）的特性数量。第三条曲线显示的是已设计（Designed）的特性数量。第四条曲线显示的是编码完成（Coded）的特性数量。第五条曲线显示的则是已经编码完成准备进行测试的特性数量。

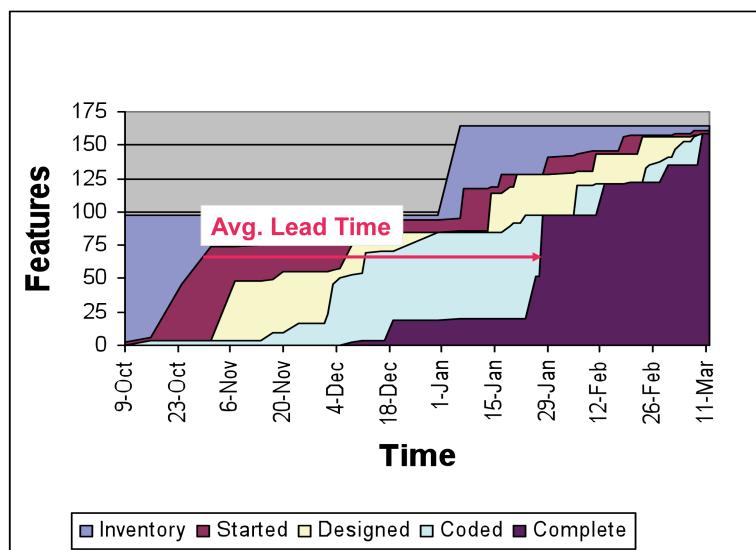


图 3.1 OTA 下载团队项目工作，从 2003 年秋到 2004 年冬的累积流图

任意一天中第二条曲线和第五条曲线之间的纵向高度，显示的是当时的在制品，即进行中工作（work-in-progress）的数量；而第二条曲线和第五条曲线之间的横向距离，显示的则是一个特性从开始到结束的平均前置时间（average lead time）。有一点需要特别说明，横向距离为平均前置时间，并不是某个特定特性的具体前置时间。累积流图并不跟踪特定的特性。第 55 个特性的工作开始的时候，可能第 30 个特性的工作已经完成。曲线的纵轴和列表中的某个具体特性之间，并没有任何关联性。

OTA 下载服务器开发团队缺乏纪律性，或者不太认可使用 FDD 方法，没有像 FDD 方法所要求的那样进行协同工作。他们为每个开发人员安排大量的开发工作。通常情况下，在任意时间点，每个开发人员手上同时会有 10 项特性开发工作处于“进行中”状态。OTA DM 开发团队则遵循 FDD 方法所要求的那样很好地进行协同工作。他们为全部的功能特性都编写单元测试。最重要的是，他们同时只进行小批量的特性开发工作，通常情况下，在任意时间点，整个团队进行中的特性开发工作只有 5~10 项。FDD 中的一个基准（benchmark）数据是，一般每个特性的代码量规模为 1.6~2.0 个功能点。

OTA 下载服务器开发团队位于华盛顿州西雅图市，从一个特性开始开发到完成开发并将之移交给位于伊利诺伊州香槟市的团队进行集成测试为止，该团队的平均前置时间为 3 个月，如图 3.1 所示。OTA DM 开发团队每个单元特性的平均前置时间为 5~10 天，如图 3.2 所示。对遗漏在系统或集成测试中的逃逸缺陷（escaped defect）进行度量后，发现两个团队的初始质量相差 30 多倍。OTA DM 开发团队初始质量达到业内领先水平，每 100 个特性仅有两三个缺陷，而 OTA 下载服务器开发团队的初始质量仅为业内平均水平，大约每个特性有两个缺陷。

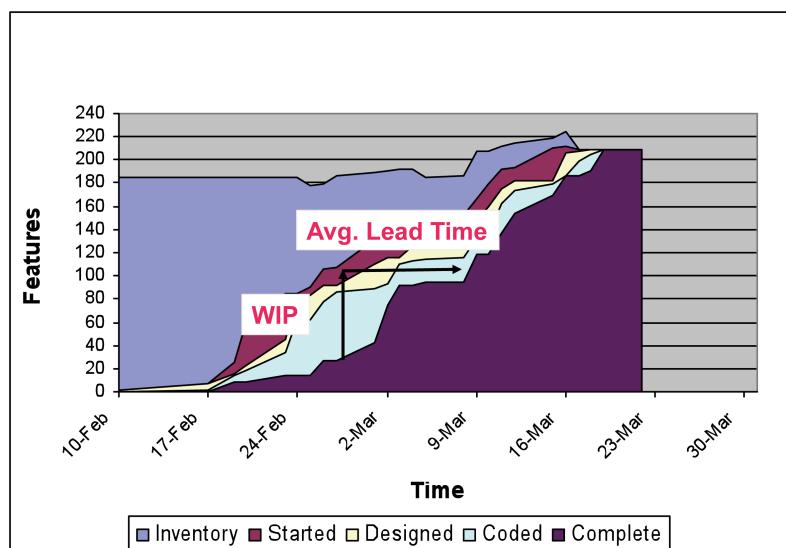


图 3.2 OTA DM 开发团队在 2004 年冬季的累积流图

仔细查看图形，可以发现，在制品数量与前置时间直接相关。图 3.2 清楚表明，当在制品数量减少时，平均前置时间也随之减少。高峰期，平均前置时间为 12 天。项目后期，随着在制品数量越来越少，平均前置时间仅为 4 天。

在制品数量和平均前置时间之间存在相关性，而且是线性相关。在制造业中，这种关系称为利特尔定理（Little's Law）。摩托罗拉公司这两支团队的资料表明，前置时间和质量之间存在相关性，前置时间增加，则质量会下降。前置时间越长，质量越会显著下降。事实上，平均前置时间增加约 6.5 倍，便会导致初始缺陷超过 30 倍的攀升。在制品数量越多，平均前置时间越长。因此，提高质量的管理杠杆点

(leverage point) 是减少在制品数量。通过对这些资料的分析，我已经将管理在制品数量作为控制质量的手段，对在制品数量与初始质量之间的关联性坚信不疑。然而，当我写作本书的时候，还没有科学的证据来支持这种基于经验观察所获得的结果。

减少在制品的数量或缩短迭代的长度，将对初始质量产生重大影响。很明显，在制品数量与初始质量之间是非线性的关系，也就是说，随着在制品数量的增加，缺陷数量会不成比例地增加。为期 2 周的迭代比 4 周的迭代好是很有道理的。为期 1 周的迭代会比 2 周的迭代更好。较短的迭代会产生更高的质量。

综上所述，仅需使用看板系统来限制在制品数量便可提升质量，这种做法更有道理。如果已经知道管理好在制品数量能够提高产品质量，那么为什么不引入明确的规则来限制在制品数量，而解放出管理人员使其可以专注于其他活动呢？

由于在制品数量和质量之间紧密相关，因此应该一起实施成功秘诀中的第一步和第二步，或者在成功实施第一步后，马上实施第二步。

谁更好？

2003 年圣诞节期间，我介入 OTA 下载开发团队，我向该团队负责人提出建议，进行中的工作（在制品）太多，会导致前置时间太长，真正完成（completed）的工作很少。我提出我的担忧，指出这样会导致质量低下。他听取了我的建议，2004 年 1 月对团队的工作方式进行了一些改变。2004 年，他们的在制品数量下降，前置时间也明显缩短。但是这个改变来得太晚，已经无法改变团队产生大量缺陷的情况。

本来该项目计划在 2004 年 3 月中旬完成，但事实上一直持续到同年 7 月中旬才告结束。OTA DM 开发团队有一半人员被调离原来的项目加入 OTA 下载开发团队修复缺陷。2004 年 7 月，尽管产品质量处于堪忧状态，该事业部总经理还是宣布产品已经开发完毕，并把该产品移交给现场实施团队。但是，后来多达百分之五十的客户因质量上的原因撤销了实施。尽管现场实施团队与产品研发团队一直维持良好的关系，但他们不信任产品研发团队的专业水平。现场实施团队认为，如果产品的质量差，那么我们也无法交付更好的东西。

很具讽刺意味的是，如果你走进西雅图 SODO¹询问那些开发人员，“这里哪些人最聪明？”他们会说是 OTA 下载开发团队。如果你再问他们，“谁经验最丰富？”也会得到同样的答案。如果你仔细看简历会发现，OTA 下载开发团队成员的平均工作经验超过 OTA DM 开发团队 3 年。从简历上看，OTA 下载开发团队成员的表现也应该比 OTA DM 开发团队成员的更好。直到今天，还有人认为他们是最好的，尽管所有的证据表明情况与此相反。

根据自身多年的管理和辅导经验，我可以这样来理解，OTA DM 开发团队的一些成员对自身的专业缺乏自信，他们担心自己的天赋比不上其他聪明的同事。但是，OTA DM 开发团队的生产效率是 OTA 下载开发团队的 5.5 倍，初始质量是他们的 30 倍。正确的流程、良好的纪律性、强有力的管理及良好的领导力，这些因素加在一起，使得两者的最终结果迥然不同。这个例子说明，并非只有拥有最好的人才能产出世界一流的成绩。敏捷社区里有些人持有这样一种信念，在敏捷开发中要想取得成功，需要的是一个由真正的好手组成的小团队。我将之称为技艺势利眼（craftsmanship snobbery）。但是，这个案例表明，一个成员能力参差不齐的团队也能够产出世界一流的成绩。

频繁发布能够建立起信任

减少在制品数量能够缩短前置时间。缩短前置时间，意味着可以更为频繁地发布可用的代码。频繁地发布代码，能够与外部团队，尤其是与市场营销团队或业务方之间建立信任。信任是一种很难定义的事物。社会学家将之称为社会资本（social capital）。他们发现，信任是由事件驱动的，小而频繁的表现（gestures）或活动，较之那些大但只是偶尔发生的表现或活动而言，更能增进信任的产生。

在课堂上，我会设问女学员与某位男士第一次约会后的感受。我会假设她的那次约会很愉快，但是之后两个星期他都没有给她打电话；而某一天，他忽然满脸歉意地拿着一束鲜花出现在她的家门口。我要她把这位男士和另外一种类型的男士进行比较。另外一种类型的男士，会在当晚约会回家的路上给她发短信说，“今晚和你在一起的时光太美妙了，我真的很想能够再次见到你。明天给你打电话可以吗？”并且第二天真的打了电话过来。猜猜女学员更喜欢哪一位？微小表现往往不费分文，较之那些大而昂贵（甚至夸张）但偶尔发生的表现而言，能够建立起更多的信任。

¹ 译注：SODO，是紧临西雅图市中心区南面的一个工业区。

在软件开发上也如此。规模虽小但是频繁、高质量地发布交付，较之规模大但频率低很多的发布，更能够在团队合作上建立起信任。

小规模的发布表明，软件开发团队具有交付能力，并能够一直致力于产出价值。软件开发团队能和市场营销团队与业务方之间建立起信任。高质量的代码发布，能够使团队与上下游合作团队，如运营、技术支持、外部工程实施和销售等之间建立起信任。

隐性知识

为什么以小批量的方式进行编码能够提高产品质量？这点其实很容易理解。在知识工作中，随着进行中工作项数量的增加，问题的复杂性也会呈指数级增加。与此同时，我们的大脑需要全力来应付所有这些复杂性问题。在软件开发中，有很多的知识迁移（knowledge transfer）和信息发现（information discovery）活动，它们是隐性的，而且都是在面对面的协作过程中发生的。虽然这些信息具备口头表述性（verbal）和可视性（visual）的特征，但它们大多以像画在白板上的草图之类的非正式形态存在。我们的大脑要全部存储这些隐性知识是不可能的，即使记住，也会很快遗忘。我们无法记得确切的细节，并会因此犯错。但是，如果团队成员在一起工作并互相帮助，通过讨论或利用群体的共享记忆（shared memory），就可以解决记忆丢失（memory loss）的问题。因此，共用一个工作空间的敏捷团队，更有可能长久地保存隐性知识。尽管如此，随着时间的流逝，隐性知识也会不断被遗忘，因此，对于隐性知识处理活动而言，更短的前置时间是至关重要的。我们知道，减少进行中的工作项，能够直接缩短前置时间。由此可以推断，如果减少进行中的工作项，则能减少隐性知识的遗忘，从而获得更高质量的产品。

总之，减少进行中的工作项，能够提高产品质量，并促进更为频繁的发布。更为频繁地发布更高质量的代码，则能增进与外部团队之间的信任。

根据交付速率来平衡需求请求量 Balance Demand against Throughput

根据交付速率来平衡需求请求量，意味着要根据交付可工作代码的速率，来设置新需求进入软件开发管道的速率。这样便可有效地将进行中工作项的数量固定在某个值。在有工作项交付后，便会从需求提请者那里拉入新的工作项（或需求）。因此，任何对新工作的优先级排序，只可能在现有工作项被交付的情境下才发生。

这一变化具有深远的影响。流程的交付速率会受限于某个瓶颈，想要知道这个瓶颈位于何处几乎是不可能的。事实上，价值流中的每个人都会声称自己已经超载（overloaded）。然而，一旦根据交付速率来平衡进入的需求请求量，在价值流中限制在制品，就会有意想不到的情况发生。只有瓶颈资源才会保持满负荷的状况。很快，处于价值流中其他环节的员工都会发现，他们有了富余能力（slack capacity）。同时，那些在瓶颈处的员工的工作会很忙，但不会过载且会被掩盖。这或许是近几年中团队第一次出现不再过载的情况，很多人也会体验到他们职业生涯中罕见的状态，终于感觉到手头有时间了。

产生富余时间

人们只有释放组织中的大部分压力，才能够集中精力准确高质量地完成工作。他们会以自己的工作为傲，越来越享受其中的感觉。那些手上有富余时间的工作者，会开始将精力投向于环境改造。他们可能会整理工作区或参加一些培训，可能会开始不断提升自身技能，改进使用的工具，改善与上下游间的沟通协作方式。随着时间的推移，一个小的改善会引发另外一些改善，人们发现，团队在持续进行改善。进而整个文化氛围都会得到改变。通过限制在制品以及只当有可用产能时才拉入新工作的做法，将能产生富余时间，这种富余时间能带来此前无法想象的改善行动。

想要进行持续改善，就要具备富余时间。为了产生富余时间，要做到根据交付速率来平衡需求请求量，限制在制品的数量。

直觉上，人们认为必须要消除这些富余时间。因此，在根据交付速率来平衡需求请求量而限制在制品数量后，人们会倾向于通过调整资源来平衡生产线（balance the line），使得每项资源都被充分有效地利用起来。虽然这么做看起来也许效率很高，也符合 20 世纪管理会计的典型做法，但它会阻碍改善文化的发展。为了能够得到持续改善，需要具备富余时间。为了具备富余时间，就必须允许价值流保持不平衡，允许有一项瓶颈资源存在。以提高资源利用率为目的的优化，是不可取的。

优先级排序

Prioritize

如果秘诀中的前三步都已经实施，那么整体应该运行得比较顺畅，应该能够做到频繁发布高品质的代码。随着在制品数量被约束，开发前置时间应该也会缩短。只有当现有工作完成、产能被释放出来时，才能将新工作拉入开发流中。这时，管

理重心应该转向优先级排序，而不仅仅只是交付的代码数量。当交付方面尚缺乏可预测性时，很少有人会去关注优先级排序的问题。当需求的交付次序不可靠时，为什么要浪费精力去排定它们的输入次序呢？在解决这个问题之前，最好将管理精力重点用于改善交付能力和交付的可预测性上。一旦能够真正做到按照需求请求的大致次序交付需求，就应该把思考转向如何对输入的需求进行优先级排序。

影响力

优先级排序本不由工程技术部门来控制，因此，也不应由工程技术管理层所掌控。要改进优先级排序，需要产品负责人、业务方或市场营销部门改变他们的行为。最好的情况下，工程技术管理层也只在优先级排序上具有一定的影响力。

为了获得政治资本和社会资本以影响变化，应该先在彼此间建立起一种相当水平的信任关系。如果不具备定期交付高质量代码的能力，就不可能建立起信任关系，因此，在优先级排序上具备影响力的可能性也会很小，也就无法进一步优化软件团队交付的价值。

目前，业务价值优化（business-value optimization）在敏捷社区中已经成为一个流行的话题，而可工作代码的生产率（称为软件开发的“速度”）已经不再是一个重要的度量指标。这是因为已交付的业务价值才是真正成功的衡量标准。从某种意义上讲，虽然也许确实如此，但是不要忽视很重要的一点，团队的能力成熟度只能循序渐进，拾阶而上，不可能一蹴而就。大多数组织都存在能力不足的情况，无法衡量和报告自身到底已交付了多少业务价值。在尝试更大的挑战之前，他们必须先建立和完善那些基本的技能。

循序渐进地构建成熟度

我认为，一个团队应该这样逐步迈向成熟。第一，要学习构建高质量的代码。第二，减少进行中的工作项数量，缩短前置时间，并频繁发布。第三，根据交付速率来平衡需求请求量，对在制品设置限额，进而产生富余时间并释放个体的创造力（free up bandwidth），促进改善行为的发生。第四，随着软件开发的顺畅运转和能力优化，通过改善优先级排序来优化交付的价值。期望一步实现优化业务价值，只是一种不切实际的美好愿望。遵循成功的秘诀（the recipe for success）并采取行动，才能逐步达到期望的成熟度水平。

消除变异性的根源，提升可预测性

Attack Sources of Variability to Improve Predictability

变异性产生的影响和如何减少过程中的变异性，这两者都是高阶的主题。为了降低软件开发中的变异性，知识工作者需要改变他们的方式，学习新的技术，并改变他们的个体行为。所有这一切都很困难。因此，它并不适合初学者或还不成熟的组织。

变异性会导致更多的在制品及更长的前置时间。第 19 章会有更详细的介绍。变异性要求非瓶颈资源具有更多的富余时间，以应对工作流中的波动，而富余时间又影响流经价值流中的工作流负载量。要想全面了解这点，要具备统计过程控制（statistical process control）和排队论（queuing theory）方面的一些背景知识，这已经超出了本书的范围。我个人喜欢唐纳德·惠勒（Donald Wheeler）和唐纳德·赖纳特森（Donald Reinertsen）在变异性和平队论方面的工作。如果读者想知道更多关于这些主题的信息，可以从他们的工作成果入手。

现在只要相信一点，即需求规模上的差异性，以及在分析、设计、编码、测试、集成构建和交付工作量上的变异性，会对流程的交付速率以及运转整个软件开发价值流所需的开销方面产生不利的影响。

然而，一些变异性的根源是由于糟糕的策略选择无意间被设计到过程中所致。第 4 章的案例研究中会重点介绍：每月的重新规划、对于估算活动的服务级别协议（service-level agreement, SLA），以及产品文本更改的优先级排序。这三个例子中的策略规则都是可以改变的。仅需改变现有过程策略规则，便能够从根源上显著降低变异性，从而提高可预测性。

3.2 成功秘诀和看板方法

Recipe for Success and Kanban

看板方法能够促进成功秘诀中所有六步的实施。看板方法能够使成功秘诀得以成功实施，而实施了成功秘诀，则能够为管理人员带来各种好处。反过来，成功秘诀也说明了为什么看板方法是一门极富价值的技术。

总 结

- ❖ 看板方法包含了成功秘诀的所有方面。
- ❖ 成功秘诀解释了看板方法为什么具有价值。
- ❖ 质量低下是软件开发中最大的浪费。
- ❖ 减少在制品即进行中的工作项数量，能够提高产品质量。
- ❖ 提高质量能够增进与下游合作伙伴如运维部门等之间的信任。
- ❖ 频繁发布能够增进与上游合作伙伴如市场营销部门等之间的信任。
- ❖ 可以通过拉动系统，根据交付速率来平衡需求请求量。
- ❖ 拉动系统能够暴露瓶颈所在，并在非瓶颈处产生富余时间。
- ❖ 对于运作良好的软件开发价值链，高质量的优先级排序活动能够使交付的价值最大化。
- ❖ 如果没有良好的初始质量，交付上也缺乏可预测性，那么优先级排序几乎毫无价值。
- ❖ 为了降低变异性而进行的改变，需要富余时间。
- ❖ 降低变异数能够减少对富余时间的需要。
- ❖ 降低变异数有利于实现资源平衡（并潜在地降低对人数的需求）。
- ❖ 降低变异数能够降低对资源的需求。
- ❖ 降低变异数能够减少看板令牌（kanban tokens）、减少在制品数量，最终体现为平均前置时间下降。
- ❖ 富余时间能够使更多的改进机会成为可能。
- ❖ 过程改进能够带来更高的生产率和更好的可预测性。

第 4 章

在五个季度内，从最差变为最好 From Worst to Best in Five Quarters

2004 年 10 月，扎格斯·杜米特（Dragos Dumitriu）还是微软公司的一名程序经理。他刚刚接管一个部门的工作。在微软公司所有的 IT 部门里，该部门的口碑最差，可谓是声名狼藉。

微软公司的“程序经理”一职，很像其他公司的项目经理，而且通常还需要承担分析和架构方面的一些职责。程序经理会被安排到一些刚刚起步的项目或产品上，负责一个特性或一组特性的开发交付。为了完成工作任务，程序经理会从开发和测试等职能领域招聘新的成员加入团队。扎格斯的团队那时负责 XIT 事业部的软件维护工作。团队的主要成员（见图 4.1）来自印度一家评级为 CMMI 5 级的供应商，其中包括 3 名开发人员、3 名测试人员和 1 名当地管理人员。他们负责为 80 多个跨职能的 IT 应用进行小规模升级开发和产品缺陷修复的工作，这些应用供微软公司全球员工使用。那时，我刚好也在微软公司工作。



图 4.1 2004 年底，该团队在印度海德拉巴 (Hyderabad)，从左边数第四位是扎格斯

4.1 问题

The Problem

扎格斯是主动申请负责带领这个团队的。在微软公司内部，该团队在客户服务上的口碑是最差的。作为变革推动者，他决心解决前置时间过长以及团队目标设置方面很糟糕的问题，但是他的工作颇受当时公司内部政治气氛的约束。该职位的几位前任程序经理，仍然是同一个业务部门内的同事，只是现在他们在负责其他项目的工作。他们也有顾虑，一旦该团队的效能得到提升，就会显得他们的能力不足。

当时，微软公司在合同中明确要求，为微软公司工作的程序员和测试人员都要遵循软件工程研究所（SEI）的个人软件过程/团队软件过程（PSP/TSP）方法。那时乔恩·德·范（Jon De Vaan）直接向比尔·盖茨（Bill Gates）汇报工作，他也是 SEI 瓦茨·汉弗莱（Watts Humphrey）的忠实拥趸。作为微软公司工程卓越计划（Engineering Excellence）的负责人，他负责指定微软公司 IT 部门及其供应商要遵循的过程。这意味着，扎格斯想要改变使用的软件开发生命周期方法（SDLC）是不可能的。

扎格斯意识到，无论是 PSP/TSP 方法，还是供应商的 CMMI 评级，都不是问题的根源所在。事实上，该小组的产出也并不比所要求的少，而且质量也很高。然而，他们的变更请求（change request）前置时间却长达 5 个月，随着请求待办项列表（request backlog）堆积的越来越长，情况就会逐渐变得不可控制。外部便会对这个团队产生一种成见，认为这个团队的组织和管理十分糟糕。因此，高级管理人员也没有意向提供更多的经费来解决这个问题。

变革受到了来自政治、财务和公司政策等多方面因素的制约。于是，扎格斯来征询我的建议。

4.2 可视化工作流程

Visualize the Workflow

我要求扎格斯画出工作流程图。他画了一幅简图来描述变更请求处理的生命周期过程，他一边画，我们一边讨论问题。图 4.2 即是他所画简图的一张摹本。图上的程序经理小人代表扎格斯本人。

请求的到达是不受控制的。4 名产品经理分别代表各自的客户并负责控制预算，这些客户是 XIT 维护的应用软件的所有者。他们会提出新的请求，其中包括遗留的

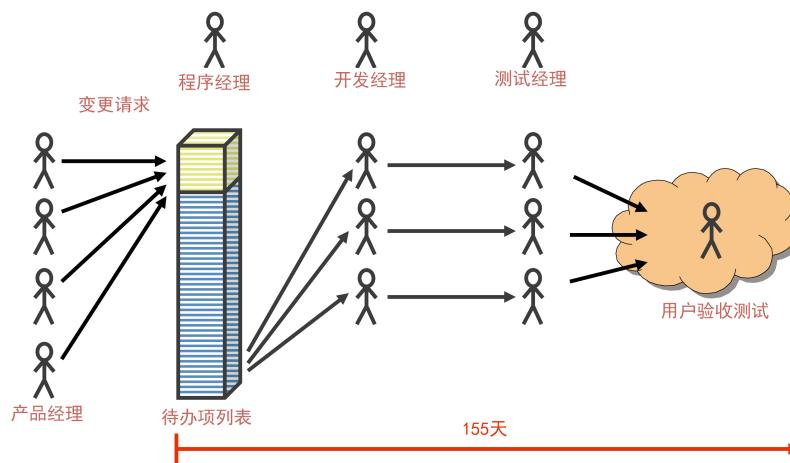


图 4.2 XIT 维护工程：最初的工作流程，展示了所需的前置时间

（在使用现场才发现的）产品缺陷。这些缺陷并不是由维护团队造成的，而是应用程序开发团队在产品开发过程中产生的。在新系统发布一个月后，这些应用程序开发团队一般都会解散，而源代码则转交到维护团队手上。

4.3 影响效能的因素 Factors Affecting Performance

当请求到达时，扎格斯会将其发送到印度进行估算。估算的规则是，必须在 48 小时内完成并返回估算结果给业务方。这将有助于业务方进行投资回报率（ROI）计算，然后决定是否要提交这个请求以进行后续处理。扎格斯每月会和产品经理及其他干系人（stakeholders）会面一次，调整待办项优先级，并根据排序结果创建一个项目计划。

那时，每月可以完成处理的请求项大约为 7 个。待办项列表中的请求项有 80 多个甚至更多，而且数目还在不断增长。这意味着，每月需要对其中 70 多个请求项重新进行优先级排序和重新制订计划，而这些请求项平均需要 4 个多月才能够处理完毕。这就是导致客户满意度下降的根本原因。不断地重新排列优先级，也意味着请求者将会屡次失望。

这些请求项是通过一个称为 Product Studio 的工具进行跟踪的。该工具的升级版本，就是后来公开发行的 Team Foundation Server 工作项跟踪系统。我在教学和咨询工作中经常见到像 XIT 维护团队这样的组织，他们拥有大量的数据，但却从不使用这些数据。扎格斯开始对数据进行挖掘，结果发现，一个请求项所需的工程处理时间平均是 11 天。但是，前置时间一般却需要 125~155 天。超过 90% 的前置时间花在了排队等待（queuing）和其他形式的浪费上了。

对加入的新需求进行估算花了团队很多精力。尽管已经声明仅是粗略量级估算（rough order of magnitude, ROM），实际上客户却仍将之视为是非常精确的估算，因此团队成员便学会了进行充分准备、谨慎做出估算的做法。每个开发人员和测试人员都会花费大约一天的时间进行估算。我们快速计算了一下，单消耗在估算上的开销，占去整体产能的 33%，糟糕月份里甚至高达 40%。这些产能优先分配用于估算工作，而非编码和测试工作上。另外，对新请求进行估算也极易导致该月份的既定计划变得紊乱。

除了变更请求，团队还要处理另外一种类型的工作，这种工作称为产品文本变更（production text change, PTC），主要是图形与文本的修改，或者一些涉及表格或 XML 文件中的数值修改。这些变更并不需要开发人员来处理，通常由业务负责人、产品经理或程序经理来完成，但是这些变更也需要通过正规的测试，因此这些工作会影响到测试人员。

PTC 工作的到达，事前一般没有什么征兆，通常而言，完成 PTC 工作会比完成其他工作或估算工作要快得多。PTC 工作通常是零星地分批抵达，它们也会导致该月份的其他计划变得紊乱（见图 4.3）。

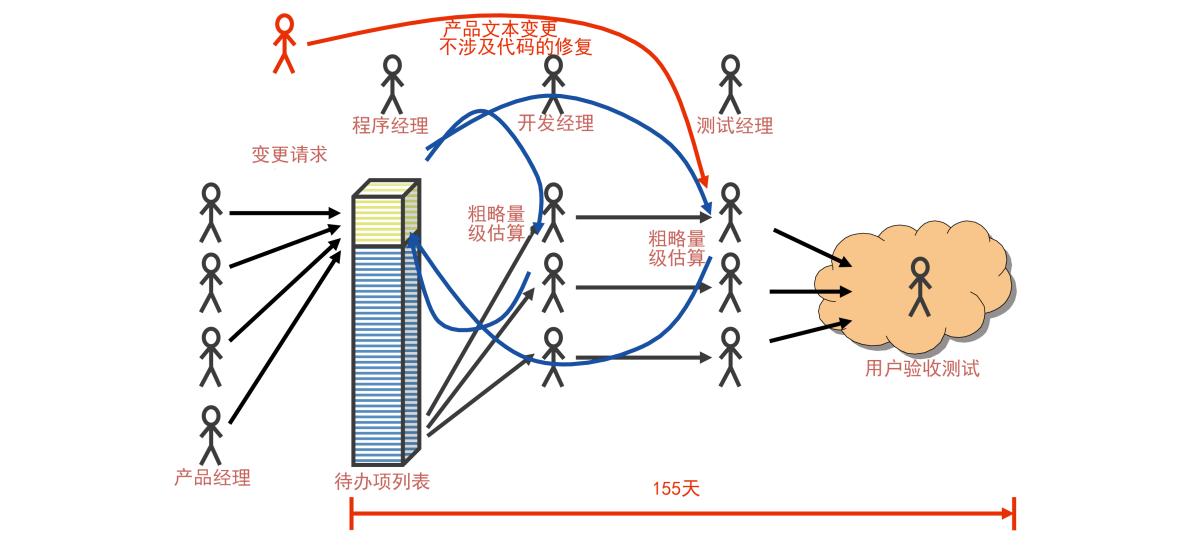


图 4.3 包含粗略量级估算 (ROM) 和产品文本变更 (PTC) 输入的工作流程

4.4 明确过程策略 Make Process Policies Explicit

该小组遵循规定的过程要求，在这个过程中，包含许多由各级管理者做出的糟糕的策略决定（policy decision）。将过程视为主导行为的一组策略，是很重要的一种视角。这些策略是由管理层来管控的。例如，使用 PSP/TSP 的策略，就是由仅次于比尔·盖茨这一级别的执行副总裁所设置的，要改变这样的策略十分困难，甚至可以说是不可能的。然而，许多其他策略，如“估算工作的优先级要高于实际的编码和测试工作”这样的策略，则是由团队内部制定的，是由直接经理们共同授权的。这些策略在最初实施的时候可能是合情合理的，而当环境发生变化时，却没有精力去检查和更新这些主导团队运行方式的相关策略。

4.5 估算是一种浪费 Estimation Was a Waste

经过与同事及经理之间的一番讨论，扎格斯决定优先实施两项管理变化。首先，团队将停止估算活动。他想回收浪费在估算活动中的产能，将之用于软件开发和测试工作。消除因估算导致的计划紊乱现象，也能提高可预测性，他希望这两者结合在一起后，能够对客户满意度产生重大影响。

然而，取消估算也存在问题。这将影响投资回报率的计算，客户也可能会因此做出糟糕的优先级选择。此外，估算还有利于方便地进行跨部门的成本核算和预算转移。估算也被用于实施一种治理策略（governance policy）：只允许将小的请求归属为系统维护的需求；那些超过 15 个开发或测试天数的较大请求，必须作为正式项目提交立项，按项目管理办公室（PMO）正规的项目组合管理流程。稍后再回头看这些问题。

4.6 限制在制品 Limit Work-in-Progress

说明：在每一时刻都保持每位开发人员只对应一个变更请求的状态。这就是一种策略选择。此后仍然可以对这一策略进行修改。将过程视为一组策略的集合，是看板方法的一个关键点。

扎格斯决定做出的另一个改变是：限制在制品，并只在当前有工作项完成的情况下才从输入队列

中拉入新的工作。他选择将在制品限定在 1 名开发人员对应 1 个请求项的水平上，对于测试人员也采用类似的规则。他在开发和测试之间插入了一个队列，以便于接收 PTC 类型请求，使开发与测试之间的工作流保持顺畅，如图 4.4 所示。可以利用一个缓冲区来平滑工作项在规模和开销上的变异性（variability），第 19 章会对这种方法进行讨论。

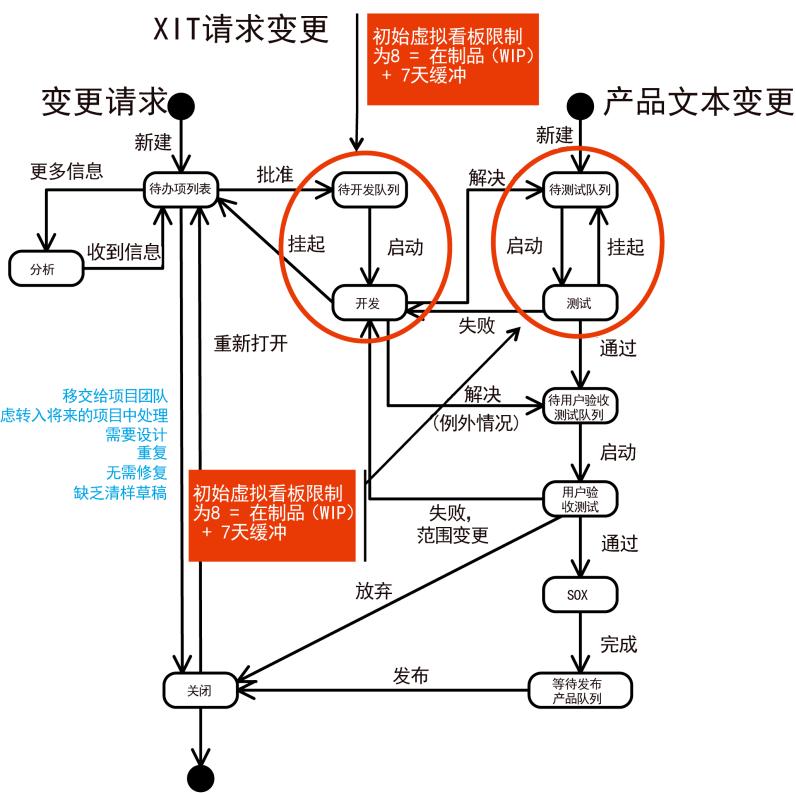


图 4.4 以限制在制品的方式形成的工作流状态图

4.7 建立输入节奏 Establishing an Input Cadence

说明：在看板方法中，节奏指的是某种活动发生的间隔周期规律。优先级排序、交付、回顾，以及其他任何循环往复发生的活动，都可以采用特定的不一样的节奏。

为了对在制品进行限制和建立拉动系统，扎格斯必须仔细考虑与产品经理之间交互的节奏。他认为，每周开一次例会是可行的方案，会议议题只专注于从待办项列表中选出待办项补充到输入队列中。一般来说，在队列中每周可能会有三个位置空出来。因此，讨论将围绕这个问题展开：待办项列表中的哪三项是你最期望在下一步交付的？图 4.5 描述了这种节奏。

扎格斯期望能够为客户提供一个确有保证（guaranteed）的交付期，在接收请求项进入输入队列开始计时的 25 天内完成交付。完成工作实际所需的平均工程处理时间

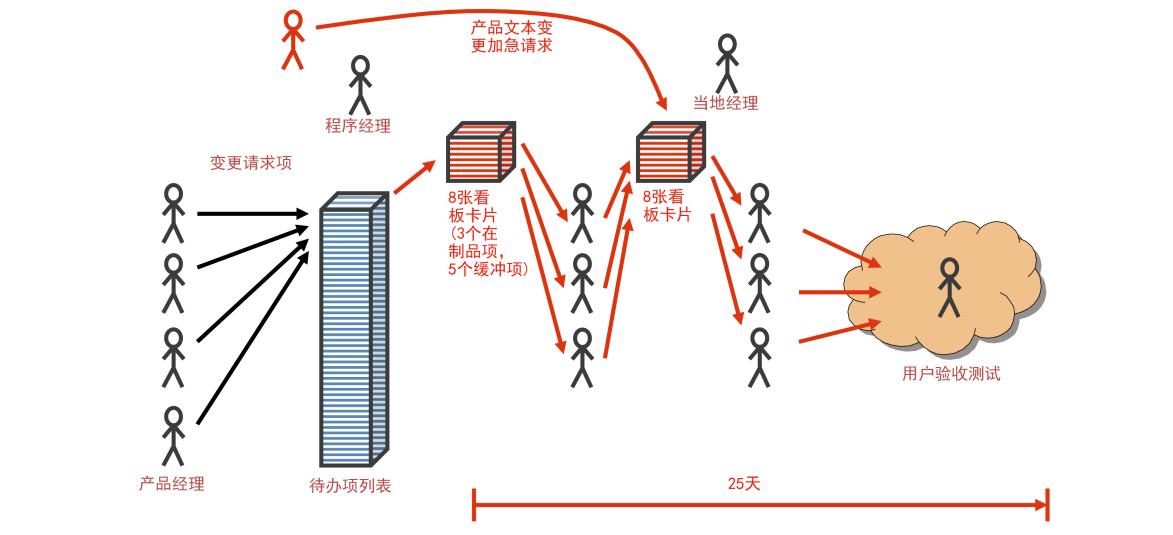


图 4.5 在看板中添加在制品限制与等待队列后的工作流程

(average engineering time)是 11 天。统计上的极端异常值是 30 天左右，但他估计这种情况很少发生；25 天听起来已经颇有吸引力，尤其当和目前 140 天左右的前置时间相比较时。他期望能够稳定地达成这一目标，与产品经理及客户之间建立信任关系。

4.8 达成新契约 Striking a New Bargain

扎格斯向产品经理提出了一项提议。他期望产品经理接受这样的方案：每周会面一次讨论优先级排序，而他会对在制品进行限制，并且团队不再进行估算活动。作为交换，他将确保交付期在 25 天之内，并且以此作为度量标准，报告准时交付的达成情况（due-date performance）。

而客户需要放弃投资回报率计算，以及基于每个请求的估算结果进行的跨部门预算转移（buget transfer）。作为交换，客户会获得前所未有的短交付期和可靠交付节奏。为了解决会计方面的问题，我们要接受将 11 天作为所有请求的平均工程处理时间。客户要接受“成本基本上是固定的”这一观点，要舍弃跨部门预算转移所基于的成本会计模式。

对这种变更可以给出合理的解释。该供应商和微软公司签订了为期 12 个月的合同，按月付款开票。供应商根据合同来分配人力资源，不管这些人是否在工作，事实上，他们都是按人头付费的。预算经费来自于 4 名产品经理，各按一定比例分配，每位产品经理摊派固定的额度费用。扎格斯保证会公平分配每位产品经理所能获得的产能。他们无须再花精力跟踪单个请求。如果他们能够接受，即他们花钱购买的是产能，而产能确实是有保障的，那么他们就可以丢弃原先基于部门成本与预算转移的偏见。可以建立一些简单的规则，来确定该将哪些请求选择出来补充进队列中以便公平地分配产能。通过一种简单的轮询方案，就可以达成这一目标。

4.9 实施变革 Implementing Changes

尽管 XIT 事业部的产品经理和许多同样从事管理工作的同事对此仍然持怀疑态度，但是大多数人都认为扎格斯应该尝试变革。毕竟，现状并不好，而且每况愈下。可以肯定的是，不管怎么变，情况都不会变得比当前更糟！是该有人去尝试变革了，大家也期望扎格斯能够带来变化。

因此，扎格斯就在团队中实施了这些变革。

变革开始产生了效果。提交的请求获得了受理，并被发布到了产品中。也实现了新约定确保交付期在 25 天内的承诺。每周例会运行也很顺畅，做到了每周一次补充满队列。与产品经理间的信任关系也开始建立起来。

4.10 调整策略

Adjusting Policies

你可能想，如果不再进行投资回报率计算，那么该如何确定优先级排序呢？实际结果表明，计算投资回报率并无必要。如果某个请求项十分重要且有价值，那么它会被选出并列入待办项队列中；如果它并不重要，那么就不会被选出。一段时间后，扎格斯意识到，需要有一项新的策略：任何处于待办项列表中已经超过六个月的请求项，都可以从列表中清除出去。如果一个请求项从提出至今已有六个月，但根据其优先级一直未能被选出进入开发队列，那么可以得出它是一个不重要的请求项的结论。如果真的很重要，该请求项仍然可以被再次提交。

说明：明确的策略、透明性和可视化这三者结合在一起，使授权团队成员自己进行决策和管理风险的做法成为可能。当管理层能够理解“过程是由各种策略构成”的时候，他们便会相信系统运行的有效性。这些策略的设计意图，便是管理风险和交付客户所期望的价值。策略明确，团队工作可透明跟踪，所有团队成员就能理解和知道如何使用这些策略。

那有什么样的治理策略，能够防止一个本该以大项目提请的需求项混入维护性质的开发工作中呢？解决这个问题的办法，就是接受一些可能会混入的大型需求。历史数据表明，这种类型的需求不足请求项总量的 2%。开发人员接到指示，要求他们对此保持警惕，如果开始处理的新请求项过于庞大，其估算开发周期需要 15 天以上，则应该告知当地经理。这样做承担的风险和付出的成本，不足以用产能的 0.5%。这是一个重大的权衡。通过取消估算，团队冒着损失低于 1% 产能的风险，获得超过 33% 的产能增量。这种新策略授权开发人员来管理风险，并且他们可以在必要时毫无顾虑地指出问题！

前两项变革在六个月内得到落实。在此期间，也进行了一些微小的调整，补充了待办项列表的清除策略，取消了与产品负责人之间召开的每周例会。整个过程运行十分顺畅，进一步地，扎格斯修改了 Product Studio 工具，实现了一项新功能：当输入队列中出现一个空闲槽位时，系统就会向他发送一封电子邮件；然后，他会发送电子邮件通知产品经理，由产品经理自己决定下一步该选择哪个请求项进入输入队列中。在开发队列出现可用资源两小时内，待办项列表中的某个请求项便会被选出并补充到输入队列中。

4.11 寻求进一步的改善

Looking for Further Improvements

扎格斯开始寻求进一步的改善。他研究了团队测试人员生产效率的历史数据，并与 XIT 部门内来自同一家供应商的其他团队进行了比较。他认为测试人员的负载过轻，还有大量的富余产能。这意味着开发人员是主要瓶颈。他决定去印度对团队进行实地考察。从印度回来后，他指示供应商在人头分配上进行一些调整。他将测试团队从 3 个人减少到 2 个人，并增补了 1 名开发人员（见图 4.6）。这在生产效率上带来了近乎线性的提升，该季度完成处理的请求项从 45 个升级至 56 个。

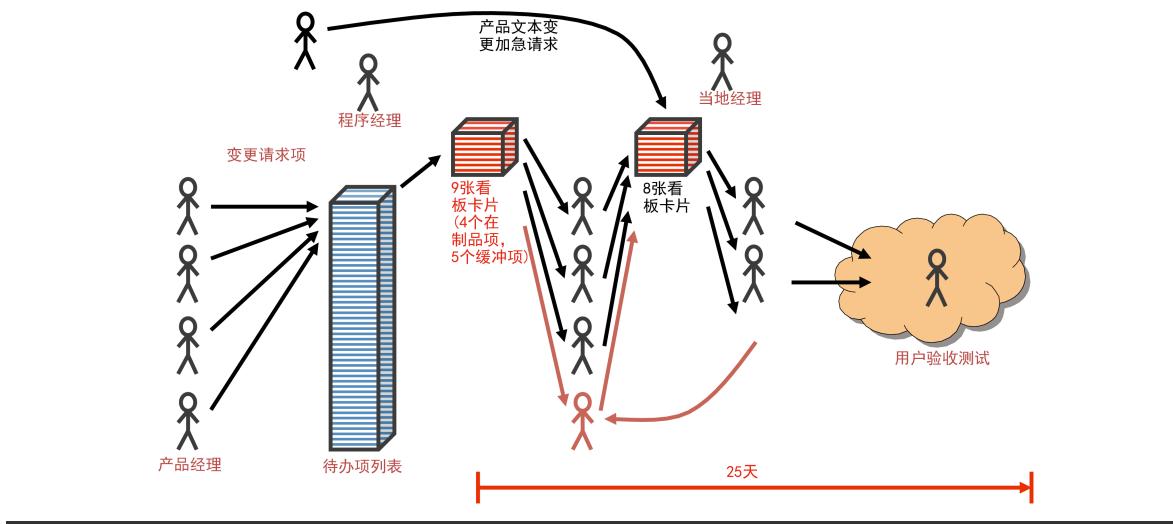


图 4.6 资源均衡化

在微软公司该财务年度结束之际，高级经理们注意到，XIT 维护（即软件维护）团队，在生产效率和稳定交付方面都有了显著提升。最后，扎格斯及其所应用的技术获得了管理层的信任。该部门分配到了增补 2 名人员的足额经费，于 2005 年 7 月新增了 1 名开发人员和 1 名测试人员。最终成果十分显著（见图 4.7）。

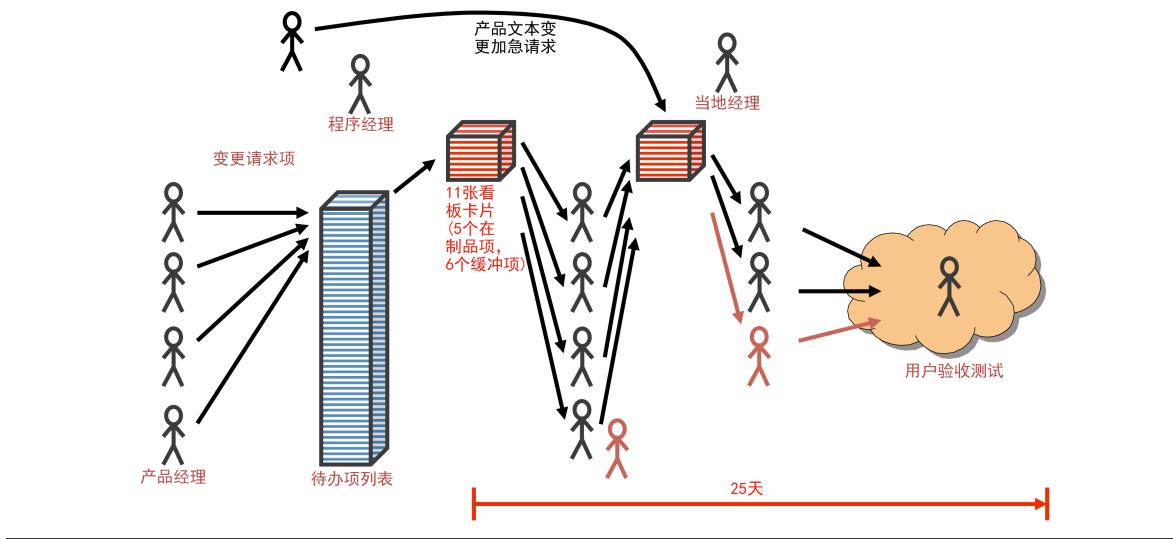


图 4.7 添加了额外的资源

4.12 成果 Results

增补的产能所提升的交付速率超过了来自业务方的请求项数量。结果如何？2005 年 11 月 22 日，待办项列表中的请求项已经全部处理完成。这时团队已经将前置时间减少到平均 14 天的水平，其中 11 天为实际的工程处理时间。25 天内准时交付的达成率高达 98%。处理完成的请求项提升 3 倍多，而前置时间则下降了 90%，可靠性大幅提高。这期间，并未对软件开发或测试过程做出任何修改。在海德拉巴工作的团队成员也感觉不到曾经发生任何明显的变动。团队也并没有改变使用 PSP/TSP 的方法，在管理条例、过程规范和供应商合约等方面也都完全满足公司的要求。2005 年下半年，该团队荣获工程卓越奖 (Engineering Excellence Award)。扎格斯获得了奖励，并开始承担更多的职责。团队的日常管理工作移交给了原来在印度的当地经理，后来该经理也被调往华盛顿办公（见图 4.8）。

之所以能够取得这些成绩，部分原因是扎格斯杰出的管理才能，但是，价值流图、工作流分析、在制品设限及实施拉动系统等诸多基本因素，是其中关键的促成因素。如果没有使用“流”的思维

范式和对在制品进行限制的看板方法，那么效能不可能取得如此大的提升。看板方法大大降低了政治风险和变革阻力，使得增量式的变革成为可能（见图4.9）。

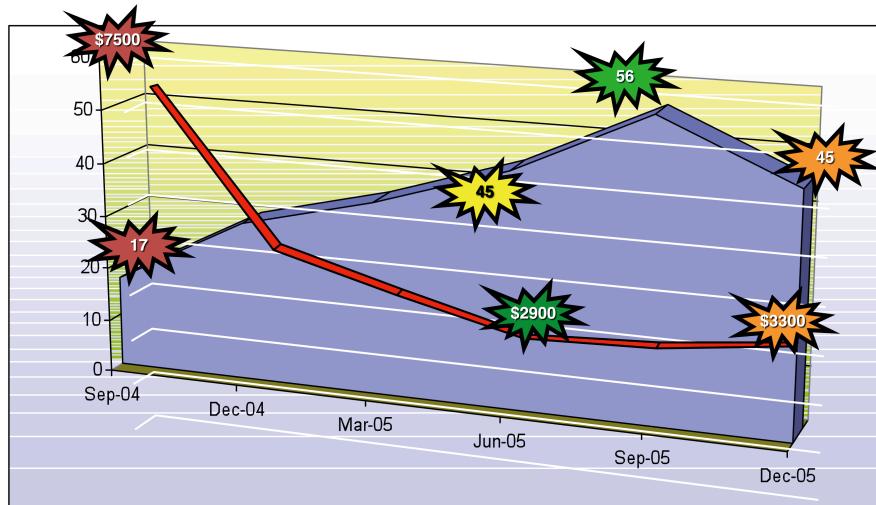


图 4.8 季度交付速率及单位成本

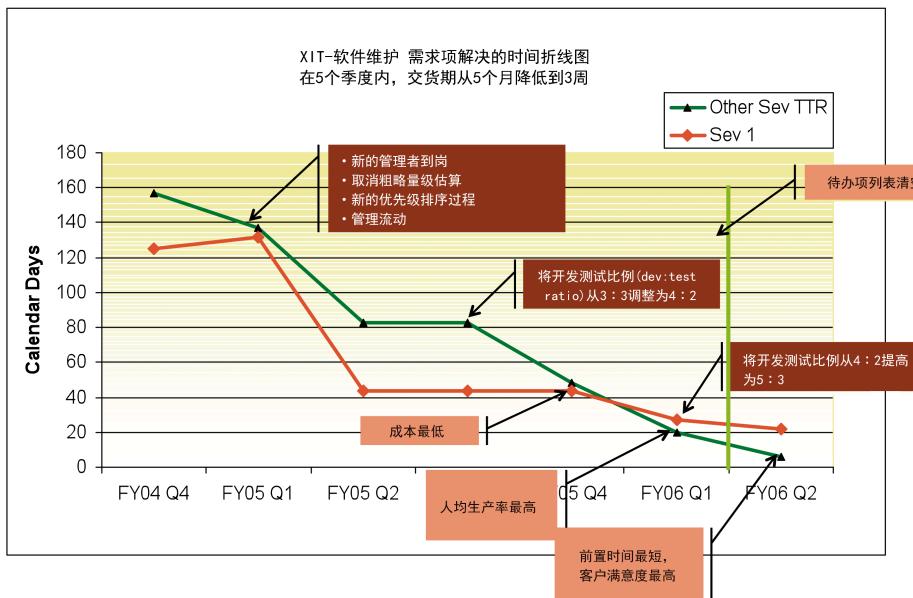


图 4.9 XIT 维护需求项解决的时间折线图，根据微软公司财政年度显示

XIT 案例展示了在一个使用离岸资源进行异地开发的项目中，实施在制品受限（WIP-limited）的拉动系统的方法，同时，在此过程中还使用一款电子化跟踪工具，以方便实施这一方法。在这个案例中，并没有可视化的故事板，很多本书介绍的看板方法的复杂特性也没有出现。尽管如此，生产率提升仍超出了 200%，前置时间缩短了 90%，可预测性获得了显著的提高，同时，政治风险和变革阻力则得以降到最低水平。对于这个案例中展现出的提升效果，其他经理自然无法再置若罔闻。

总 结

- ❖ 第一个看板系统始于 2004 年，最早是在微软的 XIT 软件维护团队中实施。
- ❖ 第一个看板系统中使用了电子化的跟踪工具。
- ❖ 第一个看板系统是在微软公司的一家离岸外包供应商团队中实施的，该供应商具有 CMMI 5 级资质，该团队在印度的海德拉巴办公。
- ❖ 应该将工作流绘制出来，以可视化的方式呈现工作流。
- ❖ 应该通过一组明确定义的策略来描述过程。
- ❖ 看板方法能够促成增量式的变革。
- ❖ 看板方法能够降低变革中的政治风险。
- ❖ 看板方法能够最小化变革过程中遭遇的阻力。
- ❖ 使用看板方法，能够发现改善的机会，而且其中不会涉及复杂的工程方法变更。
- ❖ 第一个看板系统的实施，在生产率上有了超过 200% 的提升，前置时间减少了 90%，在可预测性上也有了大幅提升。
- ❖ 通过管理瓶颈、消除浪费和降低那些影响客户期望值与满意度的变异性，产生显著改善是很有可能的。
- ❖ 变革需要经历一段时间才能充分展现效果。第一个看板实施案例在 15 个月后才完全显现成效。

第 5 章

持续改进的文化

A Continuous Improvement Culture

在日语中，“kaizen”这个词的字面意思是“持续改善”。在工作中，如果全体员工能够持续专注于提高质量、生产率和客户满意度，那么这种文化便可称为改善文化（kaizen culture）。但很少有企业能够真正练成这样一种文化。在丰田公司，几乎所有员工都参与公司的改善行动，作为持续改善的一部分，平均每个员工每年都有一项改善建议被实施。像丰田这样的公司是十分罕见的。

在软件开发领域，美国卡内基梅隆大学的软件工程研究所（Software Engineering Institute, SEI）在其能力成熟度集成模型（capability maturity model integration, CMMI）中，将最高级别的成熟度定义为优化级（optimizing）。优化级意味着该组织能够持续不断地改进产品质量和效能。虽然在 CMMI 中没有明确指出这点，CMMI 对组织文化提及得很少，但是，在一个具有改善文化的组织中，更有可能发生那些能够达成优化级成熟度的行为。

5.1 改善文化 Kaizen Culture

想要了解构建改善文化为何如此之难，首先要了解这种文化的特征。只有这样，才能讨论为什么需要打造这样一种文化，以及它可能带来的好处。

在改善文化中，员工可以获得极大的授权。个体具有行动上的自由，能够自由地去做正确的事。员工会自发地聚在一起攻克难题，讨论各种可能方案，并进行修复和改善。在改善文化中，员工不会再有恐惧感。在改善文化中，管理的基本标准是，如果实验和创新是出于改善过程和提升效能的目的，就要能够承受其可能的失败。在改善文化中，个体能够围绕手头的工作和方法，自由地（在某些限制条件内）进行自组织。整个过程清晰可控，工作任务一般是由员工自愿选择完成，而不是由上级指派。在改善文化中，具有高层次的协作和共同主导的氛围，在这种文化中，大家会超越本位观念，一起去探索如何提升业务的整体效能。在改善文化中，团队进行局部改善时，同时也会专注于系统层面，致力于提高整体的效能。

改善文化拥有很高的社会资本（social capital）。在这样一种高信任度的文化中，不论在经营决策层中身处什么位置，成员间都能彼此相互尊重，尊重每个人所做出的贡献。相比低信任度的文化，高信任度的文化往往拥有更为扁平的组织结构。高度授权促成了扁平的组织结构，让工作更为高效。因此，改善文化能够消除因过多管理层次所产生的浪费，从而降低协调成本。

改善文化的许多方面，都和现代西方文明中已经建立的文化与社会规范相左。在西方，人们从小就被教导要具备竞争力（competitive）。教育体系，无论是学术上还是体育运动上，都鼓励竞争。即使在团队运动中，也会倾向于鼓励打造英雄明星，围绕一个或两个特别有天赋的队员来构建团队。在这样的社会规范中，首先关注的焦点是个体，它依赖杰出的个体来取得胜利或保护大众免遭伤

害。因此，在工作中鼓励共同主导（collegial）的行为和系统层次上的思考与合作会十分吃力，这点也不难理解。

5.2 看板方法会加速组织成熟度和能力的提升 Kanban Accelerates Organizational Maturity and Capability

看板方法的设计初衷，是为了尽量减小变革的初始影响（initial impact），从而减小接纳变革的阻力。采用看板方法，能改善组织的文化，并帮助组织走向成熟。如果正确应用看板方法，那么组织会进化到随时准备采纳变革的状态，并善于实施变革和改进过程。SEI 在 CMMI 模型中将此称为组织创新和部署（organizational innovation and deployment, OID）的能力。事实表明，在变革管理方面具备这种高成熟度的组织，相比那些较低成熟度的组织，能够更快、更好地采纳敏捷开发方法，例如，Scrum 等。

开始实施看板方法，要先从寻求优化现有流程和改善组织文化入手，而不要抛弃现有流程转向也许会带来巨大经济效益的其他流程。这会招致一些批评，认为看板方法只是优化了一些需要改变的事物。但是，现在有重要的证据表明，在核心的高成熟度过程领域，如根因分析和解决（causal analysis and resolution, CAR）、组织创新和部署（organizational innovation and deployment, OID）等方面，看板方法能够促进组织成熟度和能力的加速提升。

当选择看板作为一种在组织内驱动变革的方法时，你肯定已经认同，对当前状态进行优化是较好的选择。因为那样做更容易，也更快捷。相比一开始就运作一场无论是从管理、工程还是名义上都要同时进行剧烈转变的变革，遭遇的阻力也会小很多。相比逐步改善现状，导入一场激进的变革要困难得多。还应该理解一点，看板方法中那些关于协作性博弈（collaborative game）方面的策略，将有助于企业在成熟度上产生重大转变。这一转变将会带来更为显著的变革，并进一步减小阻力。如果一开始就试图推进那些变革，则必会遭遇很大的阻力。选择看板方法是对组织能力、组织成熟度和组织文化的长期投资。不能简单把它视为是一种能够快速解决问题（quick fix）的方法。

案例分析：Corbis 公司的应用开发

2006 年，当我在 Corbis 公司引入看板系统时，我的目标是能够向其中导入 2004 年在微软公司 XIT 案例（见第 4 章）中已经展现出来的诸多机制。两个案例的初始应用领域是相同的，都由 IT 应用程序维护。当时我并不期待能够发生文化上或组织成熟度上的重大转变，也不期望从该项工作中演化出看板方法。

2010 年我在写作本书时，结论已经非常明确，看板方法十分适用于 IT 维护工作。但是，2006 年，对于这点我尚不清楚。看板系统的形式，对于解决系统维护工作中的机能障碍似乎相当合适。加入 Corbis 公司时，我并没有抱着实施看板（doing Kanban）这样的意图。我去那里只是期望能够提高客户对软件开发团队的满意度。令人高兴的一个巧合是，我要解决的第一个问题同样也是 IT 软件维护团队的交付缺乏可预测性。

背景和文化

2006 年，作为一家私人控股公司，Corbis 公司当时在全球拥有约 1300 名员工。它掌控许多迷人的艺术作品的数字版权，作为大约 3000 名专业摄影师的代理公司，向出版商和广告客户发放许可授权，是全球第二大商业摄影库公司。该公司同时还拥有其他业务，最有名的是许可授权业务，为名人名流代理其图片和名字的使用授权。IT 部门大约有 110 人，分为软件工程与网络运维/系统维护两大块。随着重大项目的不断投入，公司规模持续扩张。2007 年的高峰期，软件工

程部门的雇员数达 105 名，包括在西雅图的 35 名临时雇员，以及在印度钦奈 (Chennai) 的供应商那里雇用的 30 名外包人员。大部分测试工作都由印度钦奈团队完成，使用的是非常传统的项目管理方法：通过任务依赖树来规划全部事情，并由计划管理办公室来负责跟踪和推进任务的执行。这是一个文化比较保守的公司，也是一个相对保守和发展相对缓慢的产业。在项目管理和软件工程生命周期上，所采用的也是保守的传统方法。

IT 部门负责维护 30 多个各不相同的系统。有些是相当典型的会计和人力资源系统，有些是专门面向数字版权管理行业的应用，偶尔有些应用会让人琢磨不透。采用的技术、软件平台和语言分布也十分广泛。员工的忠诚度令人难以置信，IT 部门的许多人已经在公司工作了 8 年多，还有一些人的服务年份甚至已达 15 年之久。一家公司已经走过 17 个年头，这不是坏事。当前过程采用的是传统的瀑布式软件开发方式，整个 IT 部门由一个业务分析部门、一个系统分析部门、一个开发部门以及一个离岸测试部门组成，这些年这种结构已经成为一种惯例。在这些部门里有很多专家，如一些具备会计专业背景、精于财务的分析师。有些开发人员也是专家，例如，负责维护 JD Edwards 会计软件的 J. D. Edwards 程序员。

虽然这一切都不甚理想，但这就是公司的真实状况，是事物本来面貌。我加入时，公司里有些员工有不安情绪，他们以为我可能会强行推广某种敏捷方法，并强迫他们改变自己的行为。虽然这也许行得通，但过于粗暴，并且在过渡期间可能会造成严重影响。我担心这样会让事情变得更糟，员工在接受新的培训和适应新的工作方式期间，项目会陷入瘫痪的境地。考虑到因过度专业化细分造成的工种分布过于精细的现状，我也担心会发生关键人员流失这样的状况。我选择引入看板系统，让系统维护的工作回到轨道上来，然后观察期间会发生哪些事情。

建立软件维护职能的需要

预算执行委员会已经批准为软件工程部门额外增加 10% 的人头预算，用于资助软件维护团队，或者按照内部叫法称为快速响应团队 (rapid response team, RRT)。2006 年，这等同于新增 5 个人。在我来前不久，这 5 个人已经招聘就绪。考虑到所涉及系统的多样性，以及团队中高度专业化分工的现状，如果由 5 个人组成一个小组专门负责维护工作，这显然不是一种好的做法。因此，增设的 5 个人就被添加到总资源池中，其中包括 1 名项目经理、1 名分析师、1 名开发人员和 2 名测试人员。从管理角度出发，如何证明增补的 5 个人是真正投入在维护工作上而不是投入在主项目中，就成了件棘手的事情。

一种解决办法是让每个人都填写复杂的时间表，以表明团队 10% 的时间确实是用在维护工作上，但这增加了管理的负担。另外，这种办法也会严重干扰团队，但它通常会是中层管理人员应对此类问题的典型做法。另一种解决办法便是引入看板系统。

维护团队预设的期望是确保 Corbis 公司能够做到每两周便可对 IT 系统进行一次增量发布。在 Corbis 公司，一般通过主项目每三个月对主要系统进行一次升级和新系统发布。但是，随着业务的成熟，这些系统不可避免地变得越来越复杂，这种每季度进行一次主发布的节奏已经难以保持。此外，现有的一些系统事实上已经行将就木，到了不得不进行完全更换的最后关头。更换遗留系统是一个重大的挑战，而且通常项目周期会很长，需要投入大量的人力，直到新系统的功能和旧系统的差不多时，才能上线新系统而关闭旧系统（这种做法和进行系统优化相比，相差甚远，但是却屡见不鲜）。

因此，在 Corbis 公司的 IT 部门内，维护性发布正是看板可以提高业务敏捷性 (business agility) 的一个地方。

维护性小项目的形式不管用

现有系统的维护方法是以两周的跨度安排一系列短期项目来修复系统中发现的问题。有人可能会认为这是以两周一个迭代进行的敏捷软件开发，但事实并非如此。我刚加入时，碰上他们正在

协商两周后的发布应包含哪些内容，这个协商过程竟然花了三周时间才完成。因此，每次发布的前期事务成本（transaction costs）远大于其中的增值工作（value-added work）。大约需要六周，才能真正将原本规划为每两周发布一次的维护工作最终完成。

实施变革

很显然，现状已经令人无法接受，必须做出改变。现有系统已经无法提供业务所要求的敏捷性。系统维护工作为我们提供了引入变革的理想切入点。一般而言，维护工作不是关键性的任务。但是，由于业务方直接确定优先级排序，所以在优先级排序上做出的选择，对于实现短期的业务目标具有非常重要的战术意义。因此，优先级排序必须做到高度可视化。系统维护是每个人都关心并希望有效工作的区域。最后，还有一个令人信服的进行变革的理由，即大家对现状都很不满意。开发人员、测试人员和分析师需要浪费大量时间来协商工作内容，而且这种情况愈演愈烈，业务人员对最终结果也很不满意。

我们设计了一个看板系统，每两周定期进行一次发布，每次的发布时间都安排在第二周的周三下午1点钟。每周定期和业务人员一起召开优先级排序会议，时间安排在每周周一上午10点钟。因此，优先级排序以每周一次的节奏进行，发布以每两周一次的节奏进行。节奏上的这种选择，是通过与上下游合作伙伴之间共同进行讨论，基于各项活动的事务成本和协调成本所做出的决定，同时也做了其他一些调整。我们引入了一个工程就绪（engineering read）输入队列，将这个队列的在制品限额设置为5项，另外，对于整个生命周期中的“分析”“开发”“构建”和“系统测试”队列，也设置了在制品限额。对于“验收测试”“集成”和“交付就绪”队列，则没有设置在制品限额，因为它们并无产能上的约束，一定程度上也在我们直接控制的范围之外。

变革的主要效果

引入看板系统带来的效果相当显著。我们做到了每两周进行一次发布。经过约三个迭代，期间都做到了定期发布，没有发生任何意外事件。质量很好，新代码进入生产环境中后，很少甚至几乎没有发生需要进行紧急修复的现象。用于发布调度和规划的开销大幅下降，开发团队和项目管理办公室之间的争吵也几乎完全消失。看板基本达到了预期的效果。我们能够以最低的管理开销定期稳定地进行高质量的发布。发布的事务成本和协调成本大大降低。开发小组能够完成更多的工作，更为高效地向客户交付工作。

引入看板带来的意外效果

2007年1月，我们以在白板上使用便签纸的方式向开发团队引入物理看板墙。每天上午9时30分，我们在白板旁开15分钟的站立晨会。相比我们在微软公司时使用的电子跟踪工具，白板的影响更大。通过参加每天的站立晨会，团队成员时间维度上的工作流在白板上仿佛录像般清晰地展现出来。受阻的工作项会以粉红色的便签标识出来，整个团队开始更加专注于解决问题和维护流动性。生产率显著提升。

随着工作流在白板上变得可见，我开始关注流程运作的细节。随后，我在白板上做一些调整，团队中的管理人员也开始理解我所做的调整及其背后的原因。2007年3月，团队成员已经能自己做出调整。接着，团队成员个体，包括开发人员、测试人员和分析师，也开始观察和理解个中的工作原理。到初夏的时候，团队中的每个成员都能勇于提出自己的建议。我发现，团队成员会自然形成小组（通常是跨职能领域）或参与到已有的小组中，讨论流程中的问题和遇到的挑战，一旦发现可以进行的调整，就会主动实施。一般他们事后才告诉管理线上的经理。大约6个月后，软件工程团队涌现出了一种改善的文化。团队成员有了充分的授权感。恐惧感消失了。他们为自己的专业水平和取得的成果感到十分自豪，并希望做得更好。

5.3 社会学变革 Sociological Change

除了 Corbis 公司的案例，行业内也出现了其他类似案例的报道。Indigo Blue 咨询公司的罗伯·哈撒韦（Rob Hathaway）在位于伦敦的 IPC Media 公司的 IT 组织内第一个真正重现了这些效果。其他人重现了我在 Corbis 公司所观察到的由看板方法带来的社会学效应，这一点使我相信，其既非巧合，也不是因我个人参与所带来的直接结果，其中一定存在某种原因。

到底是什么带来了这些社会学变革，我思考了很多。10 年中，敏捷方法为我们带来了在制品（正在进行中工作项）上的透明度，但是相较通常所说的敏捷软件开发团队，遵循看板方法的团队能够更快速有效地形成持续改善的文化。一般情况下，向现有敏捷方法中加入看板方法的团队，会发现团队成员之间的社会资本有了显著的提升。为什么会这样？

我的结论是，看板方法不但提供了工作上的透明度，还提供了过程（或工作流）上的透明度。通过展示工作是如何从一个小组传递到另一个小组，看板为团队带来了一种可见性。看板方法使得每一个干系人都能看到他们积极行动或者消极怠惰所产生的影响。如果一个工作项受阻，而有人有能力对阻塞进行疏导，那么看板会显示这一点。比如，也许是遇上了一个需求模糊的问题。通常情况下，能够解决需求模糊问题的专家可能希望收到邀请出席会议的电子邮件。后续电话跟进之后，最终会安排一次与他们日程匹配的会议，而这次会议也许被安排到了两个星期之后。有了看板和它所带来的可见性，专家就可以看到滞后行动带来的影响，就会考虑会议的优先级，也许就会重新安排日程以便当周召开会议，而不是拖延到两个星期之后才开会去解决这个问题。

除了过程流动上的可见性，限制在制品也能够驱动大家更快捷、更频繁地进行富有挑战性的互动。团队成员很难对一个受阻工作项视而不见、自顾自地继续做其他事情。看板方法的停止生产线（stop the line）机制，能够鼓励整个价值流中的人员产生群策群力、攻克难题的行为。当来自不同职能区域、不同职位的人共同围绕一个问题合力去寻找解决办法，以维持工作流的稳定顺畅和提高系统层面的效能时，社会资本和团队的信任水平便得到了极大提升。协作得到改善，进而会带来更高的信任水平，而经由更高的信任水平，则可消除组织对变革的恐惧感。

限制在制品，加上服务类别（将在第 11 章中介绍），能够授权个体根据自身情况安排计划，而无需管理人员的监督。授权行为表明上级信任下属自身能够做出高质量的决策，从而提高了社会资本。管理者也可以从监督个体工作的行为中解放出来，将其精力专注于其他方面，如过程效能、风险管理、员工发展以及提高客户和员工的满意度等。

看板极大地提升了团队的社会资本水平。信任水平的极大改善和恐惧感的消除，能够鼓励团队成员更好地协作创新和解决问题。这一切最终迅速促成了持续改善的文化。

协作的病毒式传播 Viral Spread of Collaboration

虽然看板明显改善了 Corbis 公司软件工程部门的气氛，但在该组织外所获得的成果才是最引人瞩目的。看板是如何以病毒传播的方式改善全公司范围内的协作水平的，值得分析。

案例分析：Corbis 公司的应用开发（续）

每周周一的上午 10 点，负责协调 IT 系统运维发布的项目经理戴安娜·科洛米耶茨（Diana Kolomiyets）会组织召开管理会议，讨论 RRT 工作的优先级排序。与会者一般是负责各事业部业务运营的副总裁，他们再汇报给高级副总裁或其他高管。换句话说，副总裁向执行委员会成员汇报。Corbis 公司当时比较小，高级管理人员可以做到出席每周例会。并且，会上要做出十分重要的战术决策，因此需要通过副总裁级别管理者的管理能力和影响力来确保做出好的决策。

通常，每位与会者会在会前一周的周五收到一封电子邮件。信的内容大意是：“预计下周的队列中将有两个空位。烦请大家检查各自的待办项列表，选择候选项以供周一会议上讨论。”

讨价还价期

在新过程实施的最初数周内，有些与会者会带着谈判的预期来参加会议。有些与会者也许会说，“我知道只有一个空位，但我这边有两个小的待办项，能不能两个都处理？”这种讨价还价一般很少被接受。参加优先级排序会议还有其他成员，每人都必须遵循同样的规则。有些与会者可能会回应：“我怎么知道它们是小的呢？不能由你自己说了算吧？”或者还有人会这样回应：“我也有两个小需求。为什么不是选择我想做的这两个呢？”我将此阶段称为“讨价还价期”，它反映了发生在优先级排序会议上那种谈判式风格的协作状态。

民主期

大约过了6周，与向开发团队引入物理白板几乎发生在同一时间，优先级排序会议中开始引入一种民主投票制度。与会者是自发这样调整的，因为他们已经倦于在会议上争论，在会议上进行谈判是浪费时间。经过几轮完善后，这个民主投票制度最终落实下来。每位与会者可以为当周队列中的每个空位投一票。会议开始的时候，每位与会者都会提交一两个待办项作为候选。当候选的提交方式逐渐变得成熟时，有些人会带着PowerPoint幻灯片，有些人会通过电子表格来陈述业务提案。后来听说有些与会者会邀请同事一起吃午饭，席间对他们进行游说并达成一些交易，“如果我本周把票投给你的候选项，你下周会把票投给我的吗？”在优先级排序中新涌现的民主制度影响下，事业部副总裁层面的协作水平获得了提升，与此同时，整个公司层面的社会资本也增长了，虽然那时我们还没有意识到这点。当各事业部的领导者之间开始相互协作时，自身组织内的成员间的协作也会更紧密，他们会效仿各自的领导者。协作行为，加上可见性和透明性，能够培育出更多的协作行为。我将这个阶段称为“民主期”。

民主衰退期

虽然民主非常好，但是经过4个多月后，已经无法通过民主投票的方式产生最佳的候选项。公司付出了相当大的代价为东欧市场开发了一项电子商务功能。该项目提案作为一个主项目被提出，但其候选资格从一开始便遭到怀疑，有人质疑项目提案中数据的可靠性。经过多次努力，该功能最终获选并通过并正式实施。它是RRT系统经手处理过的大型开发项目之一，许多人被卷入其中，项目广受关注。项目推出2个月后，商业智能部门总监对其带来的营业收入进行了数据挖掘。结果发现，原始商业提案中所做的业务承诺竟然存在一处缺陷，而如果根据已经投入的成本来计算，则预计投资回收期将需要19年。由于看板带来的透明性，许多干系人都开始察觉到这一点，大家因此开始讨论到底有多少宝贵的产能浪费在这个决策上，而本该可以做出更好的决策的。于是，民主期便就此结束。

协作期

替代方法值得称道。毕竟，优先级排序委员会的成员大多数都是公司副总裁级别的高管。他们在业务方面拥有大多数人都不具备的宽广视角。因此，在会议开始时，他们会询问“戴安娜，当前交付的前置时间是多久？”她也许会回答“从开始到产品发布，这一阶段平均需要44天。”然后，他们会问：“从今天开始44天后，公司最重要的战略性业务计划是什么？”大家会讨论一会，但通常会快速达成一致意见。“是在戛纳会议期间要开展的欧洲市场活动。”“太好了！待办项列表中支持戛纳活动的有哪些候选项？”快速搜索之后，可能会产生一个包含6个待办项的清单。“那么，本周我们有3个空位，先从这6个中选出3个。其他几个下周再看。”很少再有争论。也没有讨价还价或者谈判。会议大约只需20分钟即可。我将此称为“协作期”。在我担任Corbis公司软件工程部门资深总监时期，这种状态反映了各个事业部之间所达到的最高的社会资本水平与信任水平。

5.4 文化变革也许是看板方法带来的最大好处

Cultural Change is Perhaps the Biggest Benefit of Kanban

观察这种文化变革的发展过程，观察它如何广泛地影响公司员工跟随副总裁级别的高管开始与其他事业部同事之间开展更多的协作，是一件很有趣的事情。这种变革具有十分深远的意义。因此，新任首席执行官加里·申克（Gary Shenk）把我叫去办公室，要我解释其中的原因。他告诉我，他观察到公司的资深员工间的协作水平和合作精神提升到了一个新的层次，并且以往存在敌对性的事业部间似乎也相处得越来越好了。他认为 RRT 过程对此功不可没。两年前，我肯定无法像现在写作本章时这样清晰地给出解释，但是他还是非常认同，看板系统极大地提升了协作水平，每个人都积极参与并协作，极大提升了组织的社会资本。

现在明确以看板方法（大写字母 K 开头的 Kanban）指称的实践，在改善文化方面的效果确实出人意料，它在许多方面都具有反直觉（counter-intuitive）的特点。加里·申克问道：“为什么现在不以同样的方式实施所有的主项目呢？”为什么不呢？因此，我们便着手在主项目组合中实施看板方法。之所以这样做，是因为看板方法促成了持续改善的文化，并且，这种文化变革也十分重要，所以，尽管在实施看板方法的过程中，需要在优先级排序、计划调度、报告、交付等机制上付出变革成本，但这些是十分值得的。

总 结

- ❖ kaizen 的意思是“持续改善”。
- ❖ 在持续改善的文化中，每个个体都有充分的授权感，能够毫不畏惧地行动，自动自发进行协作和创新。
- ❖ 在持续改善的文化中，无论成员身处组织结构中的哪个级别，彼此间都具备高度的社会资本和信任度。
- ❖ 工作在流行中流动，看板方法则同时为工作和流程提供了透明性。
- ❖ 过程的透明性使得所有干系人都能看到自己作为或不作为时所产生的影响。
- ❖ 当能够看到自己的影响效果时，个体会变得更乐于贡献时间和进行协作。
- ❖ 看板方法中对在制品设限的做法，能够使停止生产线（stop the line）的行为变为可能。
- ❖ 看板方法中对在制品设限的做法，鼓励集思广益共同解决问题。
- ❖ 通过共同解决问题和与外部利益相关者开展合作，团队的协作性得到了提升，最终提升了团队的社会资本水平和团队成员之间的信任程度。
- ❖ 看板方法中对在制品设限及服务分类的做法，能够授权个体主动拉动工作，进行优先级排序和计划调度决策，而无需管理人员进行监管。
- ❖ 授权水平的提升能够增加社会资本以及员工与管理人员之间的信任程度。
- ❖ 协作行为能够以病毒传播的方式扩散。
- ❖ 个体会跟随资深领导者。资深领导者之间的共担与协作行为，将影响全体员工的行为。



促进软件开发领域知识与创新的传播



中文 | 英文 | 日文 | 葡文 |

1kg 多背一公斤
.org

爱自然 | 更爱孩子

