第 1 章
Chaper 1

系统思考

快　照

西南航空是这样一家公司，它藐视在艰难的航空业取得成功的那些困难，成为其他公司效仿的楷模：一个很难复制的楷模。为什么这样说呢？因为模仿者可以照搬西南航空的系统中符合他们思维模式的某些部分，却不能照搬让这些部分协同工作而形成系统的机制。他们不能欣赏西南航空公司系统背后根本不同的思考方式：系统思考。

几十年以来，John Seddon一直在使用系统思考的方式来大幅改进服务组织。“采用系统的观点就是从外向内来思考一个组织，要理解客户的要求，要设计系统来满足客户的要求，”Seddon说，“要在这个高速变化的环境中取得控制权，就需要将决策制定和工作结合在一起（这样工人就能控制工作），并使用源自工作的测量指标……如果工人控制工作，他们就需要经理处理工人不能控制的事情，这些事情影响系统状况，即工作的方式。结果是适应性的、以客户为中心的系统。”㊀

Seddon的系统思考方法从向组织机构问5个问题开始㊁：1）意图：这个组织机构的意图是什么？2）要求：客户要求的实质是什么？3）能力：可以预测系统能实现什么？4）流程：工作是怎么完成的？5）系统状况：什么导致了系统中的浪费？我们在本章中将讨论这5个问题。然后我们介

㊀ Seddon,《Systems Thinking in the Public Sector》, 2008, 第70页。

㊁ Seddon,《Freedom from Command and Control: Rethinking Management for Lean Service》, 2005, 第101页。

绍一名领导者，他的任务就是在创建产品思想时引入系统思考，他就是产品捍卫者。在第3章中我们将再次遇到产品捍卫者，由其领导产品的开发工作。

运营航空公司的一种不同方法

空座位是浪费吗？如果你是一名航空公司的执行官，如何回答这个问题？

每个月航空交通协会都会发布美国航空公司的上座率（年度乘客里程除以提供的座位里程，即利用率的测量指标）。业务报告很快就会表扬去年上座率提高的航空公司。财务分析师坚持认为，一旦航空公司的上座率超出了收支平衡点，就会有越来越多的年收入。所以事情很清楚，空座位是浪费，除非你是飞机乘客，或者你是西南航空公司。

全满的飞机不会太舒服。很难找到地方放随身行李，现在人们带的东西越来越多，因为大多数美国航空公司都对托运的行李收费。但真正的问题来源于假日，当坏天气打乱了航线的时间表，又没有多余的运力时。太多乘客不得不在丹佛过感恩节，或在芝加哥过圣诞节，而不是在家过节。

航空公司在解决冬季暴风雪造成的问题上花费巨大，特别是上座率接近100%时。为了缓和财务上的冲击，美国的航空公司很久以前就不再为因天气而耽搁的乘客提供住宿了，但他们必须找到一种方法把这些乘客送到目的地。不方便导致乘客不开心，但大多数航空公司似乎不关心被激怒的乘客的成本。毕竟，坏天气不是他们的错。

这些年来，西南航空报告的上座率大约比类似规模的航空公司要低10%。这可能让你以为西南航空公司正处于财务困境，但你错了。Donald Converse在《Fast Company》杂志2008年的6月刊上总结了西南航空公司令人吃惊的记录[⊖]：

西南航空始建于1971年，一开始就建立了一种一致的模式，这种模式有别于惯例。在1978年，航空业取消了管理规定，从那时起，超过120家航空公司破产了。为什么在这种艰难的环境下，西南航空却继续成长和兴旺呢？值得注意的是，西南航空是唯一一家自1973年以来持续盈利的航空公司。西南航空如何做到让业务量增长139%？下面这些事实可能有助于你理解为什么西南航空创造了这个难以置信的记录：

- 公司在低费用方面一直处于业界领先水平，并占据了短线航班市场平均60%的市场份额。
- 公司每个雇员每年服务超过2400名客户——这使得西南航空的雇员是航空业中最具生产效率的员工。
- 雇员的流失率平均为6.4%——这在业界又是最好的。
- 西南航空公司一直是最佳雇主公司的前100名。
- 他们从未强制裁员，不论外部市场因素如何，例如经济衰退和高油价。
- 他们有业界最佳的行李处理记录。
- 他们的准点率记录是业界最好的。
- 员工抱怨是最少的。
- 最新的机群，最好的安全记录。

西南航空公司确定的目标是让那些负担不起机票的人有可能乘飞机。因此西南航空公司将汽车视为主要竞争对手，而不是其他航空公司[⊜]。西南航空公司从德克萨斯起步，开始服务3个城市（Houston、Dallas和San Antonio），这形成了一个三角形，大约4小时的航程，每条航线大约1小时。在创建一年之后，西南航空公司在这些航线上拥有4架飞机，但它没钱了，所以卖掉了一架飞机。然而，经理们认为取消1/4的航班会是一个灾难——这不会带来更高

⊖ Converse, “Thank You Herb!”, 2008。

⊜ 在欧洲，西南航空会与火车竞争，但美国没有匹配的火车系统。

的上座率，而会导致乘客减少。

这是西南航空公司快速周转能力的开始。地面的运营经理决定飞机可以在10分钟内清空乘客并加好油，准备下一个航班，从而使得3架飞机能够执行为4架飞机设计的时间表㊀。西南航空公司认识到，上座率没有考虑到飞机在地面上所花的时间。通过运营更少的飞机，西南航空大量节约了资本投资和人力成本。保持同样数量的航班为乘客提供了充分的选择，空座位适应了不断增长的交通量，消化了变数——边际成本很小。James Parker是西南航空公司的前CEO，他指出：㊁

> 对于西南航空主要的中短线航班结构，每个航班增加20分钟周转时间就会导致每架飞机每天的飞行时间减少2个小时。将这种效应放在450架飞机构成的机群上，每天就会损失大约900个小时的飞行时间。要知道，飞机停在地面上是不挣钱的。那样西南航空公司就得至少再多买80架飞机。每架波音737的报价大约是4千万美元，加起来就超过30亿美元。只要能有效率的周转，西南航空公司就不必花这笔钱。

西南航空意识到增加收入潜力比让每次航班尽可能多挣钱能够带来更多的盈利，同时客户也得到了更多他们想要的东西。西南航空的飞机可能不像竞争对手的飞机那么满，但飞机在空中的时间更多，这足够弥补低上座率的不足。根据一份报道，典型的西南航空737每天使用11.5小时，其他公司的平均值是8.6小时㊂。快速的飞机周转导致了高资本设备利用率，类似丰田创造的短时间准备制造。这些公司都发现了新的、违反直觉的方式，更有效率地使用资本设备和人员，同时又在面对变化的需求时保持更多的灵活性。在两个行业中，他们的方式都战胜了竞争对手采用的明显的规模经济。

既然西南航空能够比其他航空公司更快地周转飞机，相关人员也就能够更快地派往新的飞机，所以他们的生产力更高。短的周转时间是其他航空公司难以复制的，西南航空的周转时间比最好的竞争对手平均要少15至20分

㊀ 西南航空离港时乘客还没有完全坐好，鼓励大家找座位。现在不允许这样了，周转时间因此要长一些。

㊁ Parker,《Do the Right Thing: How Dedicated Employees Create Loyal Customers and Large Profits》, 2008, 第57页。

㊂ Freiberg《Nuts: Southwest Airlines' Crazy Recipe for Business and Personal Success》, 1998, 第51页。

钟㊀。短周转时间难以效仿的一个原因，就是许多不同部门必须在短时间内协作：飞行员、机组人员、备办食物人员、机舱清洁人员、检票员、操作代理、机坪操作员、售票代理人、行李代理人、货运经理人、加油员、机械师等。西南航空培养了跨越这些部门协作的一种很强的文化，来自不同区域的人自然地协助其他人完成工作。在许多其他航空公司，工作规则和测量指标系统不鼓励这样的协作㊁。

所有飞行员能飞所有飞机，所有飞机能飞所有航线

我们正乘坐西南航空的飞机从洛杉矶飞往丹佛。我们开心而惊奇地发现候机室的布置：两排舒适的座位，中间有小桌子和电源插头，就在入口处，而且是免费的！这正是我们这些步履匆匆人士在航班候机室里要寻找的东西，特别是当航班延误时。我们的航班晚点了，原因是旧金山有雾。当它到达时，我们体验到了那出了名的快速周转，登机的队伍管理得很好。但不妙的是，我们不能乘那架飞机离开，飞行员宣布它有机械故障。

我们被告知登上另一架飞机，并要求坐在同样的座位号上。我们没有登机牌，根本没有。飞机上座率大约在70%（这是西南航空的典型情况），所以大部分中间的座位都是空的，没有人太担心座位的问题。从第一架飞机的快速出口步行不远，我们就来到了第二架飞机的登机口。这架飞机是空的，队伍很短，当我们登机时，两名检票员站在门口，从一份清单上划掉我们的名字。很快我们就坐在同样的位置上，机组人员也是一样的，行李也到位了。在发现机械故障之后半小时之内，我们就乘另一架飞机离开了。

这趟旅行之后不久，我们从亚特兰大乘坐一个傍晚的航班回家，飞往明尼阿波利斯。这次是另一家航空公司，我们又遇上了雾。当夜幕降临时，我们的航班晚点了，先是推迟到晚上11:00，然后推迟到晚上11:30，然后再推迟到午夜12:15。我检查了登机口飞机到达的时间，确定我们会有一架飞机。到达的时间是11:05——这距我们的航班起飞的时间超过了1小时。

㊀ Parker,《Do the Right Thing: How Dedicated Employees Create Loyal Customers and Large Profits》, 2008, 第57页。

㊁ 参见Grittell,《The Southwest Way》, 2003, 第3章。

这肯定不是我们的飞机。简单的周转怎么需要80分钟？我们很快就发现了答案。

我们走到登机口，发现机组人员已经在那里了，他们和我们一样，已经等这架飞机等了几个小时。在传统的系统中，没有概念说一个航班和它的机组人员可以简单地派往下一架可用的飞机。每个人必须等待相应的飞机到达。我们可以看到为什么西南航空实现了更高的生产效率，它强调简单性：所有航班和所有机组人员都可以使用所有可用的飞机。

当我们要搭乘的飞机终于到达时，它似乎根本不能带人走。当然，它是全满的，更糟糕的是，它的随身行李也超多。这家航空公司收取15美元的行李费，这鼓励乘客尽可能随身携带行李。当我们登机时，同样的事情发生了——飞机几乎全满了，而且几乎每个人都带着不少的行李。它根本不能装下所有的行李。行李费用对周转时间的影响似乎是大多数航空公司的损失。你可能会想，这些行李费能否抵消飞机利用率降低所带来的损失。

Mary和Tom Poppendieck

Jim Parker是西南航空公司的前CEO，他声称西南航空公司并非真正属于航空业。公司是在客户服务行业，只是碰巧开飞机而已[⊖]。他相信客户服务行业成功的秘密就是仔细照顾好你的雇员，因为这样他们就会仔细照顾好你的客户，而满意的客户就会带来业务成功。所以西南航空公司关注3件事：创造好的工作环境，为客户提供他们真正想要的东西，确保航线总是盈利，这样它就能够长期经营。

取景框1：关注客户

如果把组织机构看做一个系统[⊜]，那么就可以用一个清晰的、以用户为关注焦点的目标来进行系统思考，从一开始就得到一些整体式的决定。开始问你自己：“谁是我的客户，他们从我的组织机构真正想得到什么？”这并不

⊖ Parker,《Do the Right Thing: How Dedicated Employees Create Loyal Customers and Large Profits》, 2008, 第65页。

⊜ 我们在使用“系统”一词时有两层意思：第一，有你的组织机构的工作系统；第二，有你交付给客户的系统。应该能够从上下文中区分它的意思。系统思考适用于这两种系统。

像听起来那么容易，因为你可能不清楚谁是真正的客户。所以，让我们从定义谁是客户开始。

谁是客户

客户是这样一些人，他们为你的系统创造的价值付钱，他们使用这些价值，支持这些价值，并提供附加的价值。还有更多的内容。如果你为一个大系统的子系统工作，就会有子系统的客户和整个系统的客户。所以清楚说明谁是你的客户可能是有点难度的。

我们建议你将问题简化，考虑你参与创建的整个系统而不仅仅是软件，以此确定你的客户。如果你要创建一个大系统的子系统，请将创建系统其他部分的人视为合作伙伴，专注于理解整个系统的客户。例如，如果你正为医疗设备程序员开发一个软件，主要客户就是那些靠这台医疗设备改善生活的人，接下来是选择、交付这台医疗设备，以及为它编写程序的医疗人员。如果你要开发的软件将使业务流程自动化，软件的目的是支持最有效率的业务过程，最合适的成功测量指标就是改进后的业务过程所带来的业务效果。

为系统付费的客户

我们想到的第一类客户通常是为系统出资的人。问问你自己：“为什么这些客户在我的系统上花钱？他们想达到什么目的？他们的主要约束是什么？”出资人通常有明确的目的：他们知道投资预期的总体结果。成本通常是限制性的约束条件，决定了投资是否有意义，有时候会有最后期限，这决定了系统是否有用。

出资人通常并不关心系统的细节，只要他们的目的达到了就行。重要的是清楚这个目的是什么，并将它与实现的细节分开来考虑。目标就是要确保出资人的目的得到实现。

使用系统的客户

使用系统的客户可能与为系统付费的客户不是同一群人，有时候这些用户有不同的目的。他们有工作要做，他们希望系统帮助他们更有效地完成工作，更舒适、更直观。正如Carl Kessler 和John Sweitzer在《Outside-in Software Development》中所说的：“如果你的产品让最终用户感觉到聪明、

高效，并能控制他们的工作，你就拥有了一个成功的产品。”⊖

支持系统的客户

当你发布一个系统时，你可能会松一口气，因为你的工作已经完成了，但对于支持系统的人来说，工作才刚刚开始。你是否知道系统的可消费性如何？也就是说，你的客户多快、多容易通过你的系统实现价值？安装一个系统或一个发行版需要哪些工作？它有多难用？需要多少培训？

你也应该理解系统的健壮性。它在正常使用时，多久会出错一次？它需要多少时间恢复？找到并消除错误的根源有多容易？失效对你的客户来说有什么代价？

从系统获得价值的客户

创建竞争优势最有效的方式，就是帮助你的客户获得成功。所以要问你自己，是否知道客户要花多长时间才能开始从系统中获得价值。你可以帮他们做些什么吗？

许多公司通过理解客户的价值链来扩展他们的业务，帮助客户的客户获得成功。例如，如果一个公司帮助他们的客户开发了一个有效的呼叫中心，或者卖了一些软件工具，提高了财务分析的效率，他们的客户就会高兴。

产品拥有者不是客户

我曾在一个公司面对着一屋子大约70个人讲课。“这里有多少开发者？”主持人问道。大约50个人举手了。“开发者中有多少人理解你们代码的目的？”只有两个人举手！

在接下来的两天，我们发现参加课程的开发者都有类似的情况。他们很少知道为什么他们要负责开发指派的功能。我鼓励他们承担责任，理解并实现工作的目的，他们应该花时间与制造工厂的生产工人在一起，弄清楚系统为他们服务的情况。后来，很多人这样做了，但生产工人认为很重要的问题从来不能进入开发者的任务清单。

⊖ Kessler和Sweitzer,《Outside-in Software Development: A Practical Approach to Building Successful Stakeholder-Based Products》, 2008, 第25页。本书中文版《成功软件开发方法 —— 由外到内开发实践指南》已由机械工业出版社于2009年出版，ISBN: 978-7-111-25793-6。

“我们是中心IT部门，”部门经理告诉我，“当我们实施Scrum时，我们决定业务单位的IT人员应该是产品拥有者。但问题是他们实际上是项目经理，所以他们通常告诉开发团队要做什么，但并不说为什么。他们关注于测量有多少新特征进入了系统，所以他们认为维护工作没有优先级，即使我们的系统让生产工人生气并常常使工厂的生产慢下来。”

产品经理和产品拥有者不是客户。他们的工作应该是连接开发团队和客户，但这里显然并非如此。相反，产品拥有者代表了开发团队向真正的客户递交产品，让开发者不用关心他们工作的目的。

后来在我们的课程中，我们设计了一个问题解决练习，两个小组独立地决定解决问题的方案，不允许完成他们知道应该完成的维护工作。两个小组对这个问题得出了相同的解决方案：他们决定每次迭代保留20%的速度，完成他们自己选择的任务，这让他们有时间修复系统，他们也为之感到自豪。

Mary Poppendieck

目的是什么

从客户的角度简单声明目的，可以神奇地澄清什么重要、什么不重要。例如，西南航空公司的每个人都知道，公司的目的是让每个人觉得飞行舒适并有乐趣。这意味着费用必须低，麻烦必须最少。这个目标在公司的各个层面上指引着战略计划和日常决定。

当确定了客户之后，就需要从客户的角度来清楚地理解组织机构的目标。创建一个简洁、干脆的声明，说明组织机构认为什么对成功最重要。这是你的工作系统必须交付的东西。注意，你的目标可能不是开发软件。很少有客户想要软件，客户希望解决他们的问题。如果客户可以不用软件就解决他们的问题，他们会很高兴。

作为一个例子，让我们来看看如何来描述“产品拥有者不是客户”这个小故事中开发团队的目标。在这个例子中，首要客户是用户，即产品的操作者。另一个重要客户是出资人。第三类客户是支持产品的运营人员。开发团队的目标是提供生产信息系统，让产品操作者的工作更容易，同时提高制造

工厂的品质和生产效率。

让我们来看另一个例子。Werner Vogels是Amazon.com的CTO，他为他的IT部门定义了目标：为Amazon.com及其业务伙伴提供高可用的、高可伸缩的技术平台[⊖]。所谓“可用”，Vogels指的是购物者总是可以浏览网站，将商品放入购物车，即使系统的其他部门不能工作。所谓“可伸缩”，Vogels指的是应该实现正面的规模经济，而不是负面的，并且伸缩性应该根据需要而获得。在过去几年里，追求这一目标已经使Amazon.com变成了一个令人印象深刻的技术提供商和零售商，同时Amazon的IT部门已经成了真正的利润中心。

什么是客户要求的本质

建立组织机构的系统观点的下一步，就是理解客户对你提供的产品和服务的要求模式。客户的要求有两种形式。第一种是要求产品和服务能提供价值。第二种是在产品或服务不能满足客户的期望时，要求得到失效补救。也就是说，要求修复坏了的、不够的、难以使用的、难以配置的、难以修改的东西。

失效要求

失效要求是对造成失效的组织机构的资源提出要求。例如，支持电话几乎总是失效要求。它们可能由于用户不清楚系统的某个方面而引起，或者由于系统完全不能操作而引起。

变更请求也可能是失效要求。例如，一项变更请求可能来自于你不能理解客户，或你对于客户真正想要的东西的不成熟的决定。即使你交付了客户明确要求的东西，但当他们看到的时候可能会意识到，这并不能解决他们的问题。从客户的视角来看，这种情况下的变更请求是失效要求。

失效要求的一种隐含形式就是因技术债而对资源提出的要求：例如你决定忽略的缺陷、混乱的编码实践、重复、缺少有效的测试自动化、紧密耦合的架构、多个代码分支，总之是让你难以响应变更请求的所有事情。当客户要求变更时，所有这些事情都对你的能力提出了更多的要求，即使变更本身

⊖ Vogels, “A Conversation with Werner Vogels,” 2006，也可以观看关于Vogels的视频资料, “Availability & Consistency,” 2007, 2008。

是一项价值要求。

来自你的系统的支持团队的要求通常是失效要求。如果你的产品难以与其他系统集成，版本间的数据库迁移是一场噩梦，或者间歇的死锁让数据中心宕机，那你就有了严重的失效请求。即使你的软件准确地按照它的设计工作，但如果它给操作者和支持团队带来问题，那么你和他们都将浪费宝贵的时间。

什么是失效要求

我们曾向一个管理团队解释失效要求。我问道：“支持经理在吗？”“是的，他在。”

“那么你认为你的组织的要求中有多少属于失效要求？”

“大约95%。”他很快回答道。

当天的晚些时候我为一大群人讲课，听众提了一个问题：“你说我们应该能够快速更新代码。但如果许多客户都有我们软件的定制版本，我们怎么做得到？”

我突然明白了技术支持问题的规模。“听起来你们有大量分支的问题。”我答道。确实如此。分支和支持策略在组织机构中制造了不断增加的失效要求。

Mary Poppendieck

寻找失效要求的目的是尽可能多地确定失效请求，因为失效请求是一种浪费，本来你可以做其他事。在一开始，你应该预期到许多失效要求。要欢迎这种情况，不要为此惩罚任何人。失效要求是伴随着你的工作完成而产生的，揭露它们让你有机会实现重要改进。你的首要目标就是找到并消除失效要求，从而就有更多的时间来添加其他的价值要求。

当你确定了失效要求之后，计算组织机构的请求中失效请求的百分比有多少。可能这个数字很大，我们预计它在30%至70%之间。想想看，如果1/3的要求是失效要求，而你可以消除这些失效请求，那么组织机构的能力就能提高50%。消除失效要求为提高生产效率提供了巨大的机会。在第4章中，你会看到如何发现原因和如何减少失效要求的数量。

当然，失效要求不会在一夜之间消失，只要它存在，你的下一个目标就是尽可能快地消除每个问题。John Seddon说，“作为一条经验法则，从客户的观点来看，‘问题-修复’系统中，端到端的时间几乎总是基本的测量指标。”[⊖]问题-修复系统从苦恼的客户开始（有些东西出问题了），到问题解决（修复）结束。对所有你决定修复的失效要求而言，基本的、以客户为中心的测量指标就是从请求到解决的时间。很清楚，你的第一项优先任务应该是消灭失效要求，但如果有失效要求，你就要将客户的痛苦降至最小，尽可能快地修复问题。

价值要求

组织机构的首要要求应该是价值要求。这类要求的形式可能是从客户的角度请求增加价值，或者是满足客户还不知道的、未满足的需求，但你通过产品和服务而发现并满足的要求。当价值要求得到满足时，通常就会为你的组织机构带来收入。

我们所知道的几乎所有的软件开发组织，收到的服务要求都超过其能力，所以一个好问题就是：“收到的价值要求的哪一部分是我们的组织有能力满足的？”如果价值要求超出了你的交付能力，可以采取两个步骤：1）专注于消除失效要求，这样你就有更多的时间来处理价值要求；2）评估如何过滤价值要求，决定接受哪些工作，拒绝哪些工作。

当你明白你的组织正经历的价值需求的规模，知道如何定量化，知道可以接受的百分比时，下一步就是对要交付的价值建立以客户为中心的视图。客户真正看重的价值是什么？当你理解客户的价值后，你就可以建立以客户为中心的测量指标，让所有人都关心改进客户的结果，而不是满足内部的目标。如果你为交付的客户价值提供可视的方式，开发团队就可以独立而有创造性地工作，为客户提供更多想要的东西。

下面是以客户为中心的测量指标的一些典型例子：

1）产品开发的上市时间（针对整个产品）。

2）针对客户请求的端到端的响应时间（从请求到解决的时间）。

⊖ Seddon,《Freedom from Command and Control: Rethinking Management for Lean Service》, 2005, 第106页。

3）产品在市场上的成功（盈利能力、市场份额）。

4）新系统带来的业务好处（可以测量的业务改进）。

5）从交付到客户获得价值的时间（可消费性）。

6）（发布前）漏掉的缺陷的影响（客户宕机时间、财务上的影响）。

取景框2：系统能力

当你理解了你的客户，理解了向他们交付价值意味着什么之后，下一步就要弄清楚你交付这种价值的能力。你现在和将来需要怎样的能力才能在市场上取得成功？有能力交付客户今天想要的东西是很重要的，但也有必要看到未来，确保今天的努力将使你的组织机构迈向明天的成功。实事求是的长期计划和风险评估，包括理解当前和未来的部分环境，是持续成功的重要因素。如果个人和团队打算利用他们的创造力来决定今天做什么重要，那么他们既需要理解当前客户的需求，也需要理解未来的挑战。

什么是系统预计能达到的能力

要改进能力以满足客户的需要，就要从理解你当前工作方法的实际工作方式开始，当前它们的交付能力如何。你需要理解你的能力、客户的需求、竞争对手的能力之间的差异，既包括现在，也包括未来。

理解能力

如果你想知道你的组织机构表现如何，不要只看几个数据点，而要看数据随时间变化的情况。所有的系统都会表现出涨落，只看几个数据点不能告诉你系统中存在多大的涨落。实际上，随机的数据点会让你对目前的能力产生扭曲的看法。

让能力可视化有一个好方法，就是创建一个时间序列图。让我们假定你想了解快速响应某种需求的能力，例如，来自客户的紧急需求。当需求到达时，给它一个时间戳，当需求完成并关闭时，用关闭的时间减去到达的时间，得到端到端的响应时间，然后在图上标出这些时间点。紧急小需求的响应时间的时间序列点图看起来可能如图1-1所示。

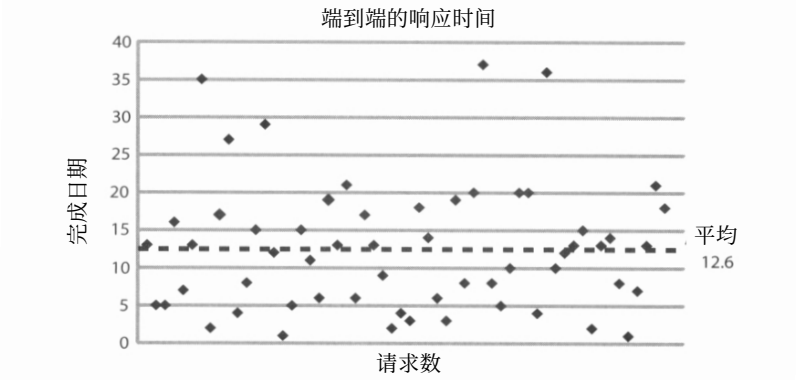

图1-1　紧急小需求的端到端响应时间的时间序列图

你对这类图的第一反应可能是寻找几项需求长时间延迟的原因。你不必到别处去找原因 —— 你的工作方法就是原因。这张图表明你的工作系统目前的交付能力。寻找替罪羊或设置更为激进的指标都不能令情况好转。你当前的工作方式产生了图中的结果，如果你不喜欢这样的结果，就必须改变工作的方式。

戴明（W. Edwards Deming）曾致力于向工业领导者宣传对涨落的理解，但我们认为他的意思常常被曲解。我们经常看到有人努力消除每个过程中的涨落现象，但却没有意识到这样的努力常常使事情变得更糟。当你试图消除过程中的正常（常见原因导致的）涨落时，过程实际上会变得更糟，而不是更好。戴明指出几乎所有的涨落（可能是95%）都是常见原因导致的（它是系统固有的），消除它的唯一方法就是改变工作的方式。他的结论是，绝大多数涨落是管理问题。所以，如果你在寻找替罪羊，那就该照照镜子。

正面强化还是负面强化

培训的一种常见方式就是当某人做某事特别好时，就给予正面强化，因为人们认为这将导致更多同样的行为。

我们听到一个飞行员培训中心决定试试这个理论。他们仔细记录下飞行员在培训时的正面强化和负面强化的结果。让研究者吃惊的是，他们发现每当受训飞行员得到正面强化时，表现就会变得更糟，而在得到负面强化时，表现总是会改进。培训中心得出结论，对糟糕的表现给予负面的反馈是一种有效的培训技巧，对好表现的正面反馈应该避免。

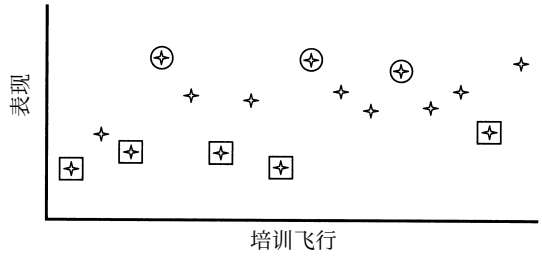

图1-2 飞行员在培训中的表现

这项调查有什么错误？请考虑飞行员表现的时间序列图，如图1-2所示。

这个时间序列图展示了表现中的预期涨落 —— 涨落是随机原因导致的，基本上与飞行技术无关。假定飞行员在高点（画圆圈的点）得到表扬。当然，下次表现就会出现在较低的地方。如果飞行员在低点（画方框的点）得到训斥，表现似乎就会改善。

这些证据不能说明正面强化和负面强化的作用，但却清楚地表明了收集数据的人不理解涨落。

Tom Poppendieck

系统需要实现什么

改变工作方式的第一步已经完成了，就是理解你需要什么来获得短期成功和长期成功。是否有竞争对手能比你更好地向客户交付价值，或者存在将来出现这种竞争对手的威胁？你是否能通过重要的改进获得有意义的比较优势？如果照目前这条路走下去，你的表现是否会变坏？你的架构是否支持可持续的增长？你是否欠下了一些技术债，让将来的发展变慢？

当你理解了现在成功和将来成功需要的东西之后，你就可以创建一个长期的目标，让每个人都清楚地知道，组织机构要想现在和未来获得成功，都需要做些什么。这会让个人和团队平衡短期、局部的决定与长期、全局的使命。

不要设置目标

你不应该设置目标。你当前的系统就是这个样子，目标不会改变它的能力。你要测量系统的能力，而不要规定它的能力。正如戴明所说的，“ 如果你们有一个稳定的系统，那么设定目标是没有用的。你们会得到系统能交付的

东西。超出系统能力的目标是不会实现的。如果你们没有稳定的系统，那么设定目标是没有意义的。没有办法知道系统会生产什么：它没有能力。”㊀

可以这样来考虑。假定你设置的目标超出了系统的能力，这个目标是当前的工作过程不能实现的。因为目标带来很强的动机（没有争论），所以人们有三个选择：1）重新设计工作系统；2）扭曲系统（例如，忽略一些缺陷）；3）欺骗（玩弄）系统以达到目标㊁。如果人们不知道如何重新设计系统，他们就只有选择后两项：扭曲或欺骗。

假定人们确实知道如何重新设计工作。根据我的经验，重新设计工作带来的结果有可能比任意指定的目标要好得多，但是人们有什么动机来超过这个目标呢？类似的，如果你设置的目标低于当前系统的能力，就可能得到低于系统交付能力的结果，因为人们没有动机去超越这个目标。

目标会跑偏

管理者们肯定听到过，他们应该设置有挑战的目标，目的是为了激发最好的表现。但在《Goals Gone Wild》㊂一文中，来自4个商业学院的研究者对这种做法提出了质疑。他们同意目标是很强的激励，太强大了，以至于常常带来严重的副作用，可能带来有危险的产品、不道德的行为、严重的低效系统，或减少学习。

可能有太多的目标，从而导致人们选择他们喜欢的去实现。或者目标太具体，从而导致了不够优化的行为。或者目标太有挑战性，导致人们不计代价去实现目标——这引入了不必要的风险、欺骗或扭曲。该文的作者发出警告：“目标可能由于太局限、不道德行为、过度的风险、减少合作、降低内在的动机等原因，在组织机构中导致系统性的问题。”他们提出，“在组织机构中设置激进的目标将鼓励组织的风气转向不道德的行为。”

毫无疑问，目标能影响行为，但它们只是生硬的工具。所以要仔细思考你想要的东西，你才可能会得到它。

Mary Poppendieck

㊀ Deming,《Out of the Crisis》, 2000, 第76页。

㊁ From Seddon,《Systems Thinking in the Public Sector》, 2008, 第97页。

㊂ Ordóñez等, “Goals Gone Wild: The Systematic Side Effects of Over-Prescribing Goal Setting,” 2009. 也可以参见Bazerman, “When Goal Setting Goes Bad,” 2009。

目标是一个预先确定的绩效水平，要求人们实现。如果严重偏离目标（或计划），通常要求给出解释。目标通常与激励有关。根据偏离目标的程度，人们得到奖励或受到惩罚。

基于绩效目标的激励系统假定，只要人们更努力，他们就可以实现更好的结果——因此目标表达了一种隐含的假定，即人们目前没有尽最大的努力。让个人和团队努力交付卓越的客户成果并相信他们会尽全力，这样不是更好吗？

小心使用相对目标

并非所有的目标都是基于预先确定的目标，目标可以基于相对的测量。例如，目标可以基于竞争对手的绩效，同事的绩效，或者过去的绩效。相对绩效目标没有试图为将来的绩效设置固定的水平，而是看实际的绩效与其他绩效相比的情况。在大多数运动竞赛中，输赢取决于相对表现：游泳运动员只要比其他7个选手快就能获胜，棒球队只要比对手得分多。

相对目标可以带来很强的动机，而且也比目标或计划具有一些优势。相对目标不是基于对未来的预测，也不会鼓励人们扭曲或欺骗系统，而且，也不会成为绩效的最低要求。另一方面，相对目标可以刺激竞争对手破坏其他人的绩效。因此在同事之间比绩效时可能对公司造成破坏，特别是当奖励与评比挂钩时。但是，如果竞争是友好的，绩效奖励是所有竞争者平等共享的，在同事间评比绩效的破坏效果就能得到缓解。

挑战：从未来倒推

要让人们积极参与，让组织变得更好，也许最好的方法就是创建一个长期愿景（生存下来所需要做到的目标），给每个人一个挑战，让他们帮助组织朝着需要达到的目标前进。挑战个人和团队，让他们改进完成工作的方式，重新设计他们的工作方法。挑战他们，让他们开发出一些系统，提供清楚的业务价值。挑战他们，让他们创建更好的产品，在市场上获得成功。

假定你希望减少产品发布前那一段“艰难时刻”。目标可能是“把测试的时间减少一半！”如果足够重视这个目标，测试的时间可能就会减少一半，但通常代价就是使更多的缺陷进入了产品之中。

更好的方式是对团队提出挑战，让他们重新设计开发工作的方式，这样缺陷在进入代码之后能够尽快被发现并修复。在这项挑战面前，团队会放弃

关注交付了多少特征，开始思考如何确保交付的特征没有缺陷。团队会引入自动化的测试和持续集成，并采取办法，在检测到缺陷时马上停下来修复缺陷。如果执行得好，这种方法就可以创下纪录，大幅减少后期测试的时间（也许超过一半），同时提高了品质和生产效率。但这样的结果只有在团队致力于改进工作方法的能力，努力编写无缺陷代码时才能实现。命令将测试时间减少一半，是很难达到这种结果的。

挑战不同于固定的绩效目标：

1）挑战不一定是SMART⊖的，它们可以是开放的，以客户为中心的，目的是激发热情和自豪感。

2）挑战体现了信任，相信个人和团队是聪明的、有创造力的、能够独立思考，相信他们能够尽全力达成组织机构的目标。

3）挑战来自于长期愿景，即要想取得长期成功需要做什么，它包含了足够的信息，让个人和团队可以独立行动，同时又确信他们的工作能对实现愿景做出贡献。

4）因此挑战是从未来倒推的，而不是预测未来。

取景框3：端到端的流程

通常，提供客户价值是通过一系列独立的输入-处理-输出步骤来实现的，而不是通过组织机构的一个集成的工作流。但是，如果没有人从端到端的角度来看待客户问题或产品是如何通过工作系统的，那么客户问题就可能卡在部门之间，辛苦得到的知识在工作传递时消失，客户真正注重的价值一路损失殆尽。形成对工作通过系统时端到端工作流的理解，是系统思考的基础。

要评估通过系统的端到端工作流，一种方式就是画出流程图（也被称为价值流图），它体现了产品或客户问题通过你的系统时的端到端的流程。画这些图有两个理由，它们能帮助你：

1）发现失效要求的原因，所以可以消除失效要求。

2）找到工作流中的浪费，所以可以消除浪费。

⊖ SMART的目标是具体的（specific）、可测量的（measurable）、可达成的（attainable）、实事求是的（realistic）、及时的（timely）。

消除失效要求

失效要求就是浪费。所以最好问一问：“为什么这个过程会产生失效要求？可以改变什么来防止失效要求？”一般来说，你的偏见会让你难以创建失效要求的流程图，因为那张图中的每项活动都可能是浪费。但有时候，失效要求的流程图能够帮助你发现如何减少浪费，并最终消除浪费。

失效要求的流程图

我们曾在一家Web门户公司画流程图，针对的是关键缺陷解决过程。小组希望从关键缺陷报告到达开发团队开始画这张图。但关键缺陷的解决是一个破坏 - 修复问题，所以流程图应该从客户发现某些东西遭到破坏时开始（参见图1-3）。

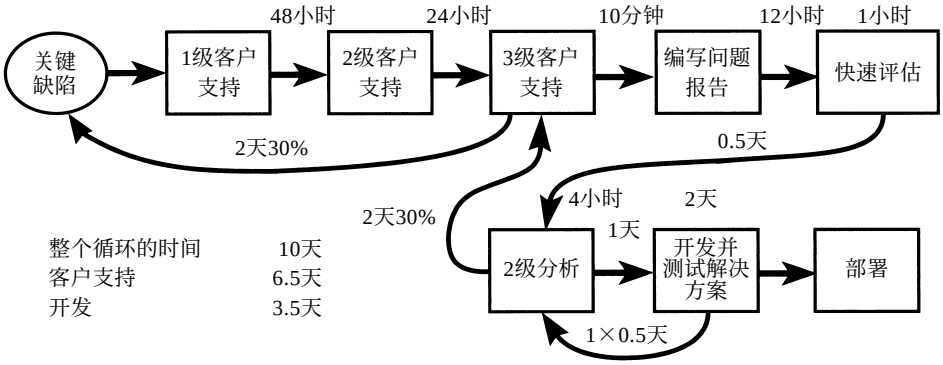

图1-3 关键缺陷解决流程图

“我是你的一位客户，”我这样说是为了让流程图从正确的地方开始。“那么当我发现了一个关键缺陷，我该怎么做？”

“打电话给1级客户支持。”

“好的。我知道情况会是怎样的：他们告诉我肯定做错了什么事情，你们的系统不可能出错。那么我需要花多少时间才能让他们确信这是你们的问题，而不是我的问题？”

“可能要几天”回答有点胆怯了。接下来我的问题进入到2级客户支持，一天以后我的问题进入到3级客户支持。通常此时没有足够的信息，所以他们有30%的可能性会回来找我。这可能要2天。

一天以后，我的问题到达了开发团队，他们马上查看了问题报告。有

30%的可能性他们会回到3级技术支持，这又会花2天的时间。但通常（30%的可能性）3级客户支持需要回来找我，询问开发团队提出的问题。需要花2天联系我，再花1天将答案告诉开发团队。

当问题清楚以后，需要4小时来开发解决方案，需要1天等待测试，测试又会花2天。一般还需要半天来修复测试中发现的问题。最后修复得到部署。

所有时间算起来，客户支持过程大约是6.5天，开发大约是3.5天。所以平均要花10天来解决我的问题。

“假设6.5天消失会怎样？”我问道。

“你是说客户直接打电话给开发人员？”开发副总裁吓了一跳。看起来这不是一种有效率的方法。

但当时有人从房间背后说话了。“我们曾那样做过，”她说，“几年前，当我在另一家公司工作时，我们当时很疯狂，我们需要试一些东西，所以我们让CEO允许我们跳过客户支持，效果很好。”

“谁接电话？”开发副总裁问道。

“开发人员，轮流接。”她回答道。

“那需要花多少时间？”他的怀疑很显然。

“好吧，我们做了很多试验，很快建立起2条规则：1）在发布后，立即建立一个团队负责接电话。2）开发者需要创建一个知识库报告，让每个打电话的客户能够访问，然后才能关闭请求。

“有了这两条规则，我们对结果很吃惊。我们有800名开发者。在改变过程之前，60%的开发时间用于修复关键缺陷。在改变了过程之后，20%的开发时间用于修复关键缺陷。”

通过让开发者和客户直接联系，这个公司实际上获得了320名有经验的开发者，免费的。

Mary Poppendieck

要记住，所有处理失效要求的时间都是浪费。所以通常最好是创建一个流程图，让它从产生失效开始，而不是创建处理失效的流程图。你的目标不是要更有效地处理失效要求，而是完全消除失效。在为失效要求创建流程图

之前，问一问自己，“是什么导致了开发过程产生了失效请求？如何改变这个过程，从而防止（或大量减少）失效请求？”

特别是，我们建议你在为处理变更批准过程创建流程图之前努力思考。客户通常不会认为变更批准系统是价值，而会认为需要的变更是价值，但要求批准进行变更的过程让人恼火。如果客户认为你的变更请求批准过程是讨厌的东西，这个过程就是处理失效请求。不要改进你的变更批准过程，要考虑为什么要有这样一个系统。在开发的系统有一个变更请求批准过程，通常表明开发过程不能吸收新的思想，或者处理客户场景中的典型变化。也许你的决定太快，或者等待太长的时间才能得到反馈，或者没有在计划中留出足够的空间来接纳反馈。也许你的客户参与和协作是无效的。问一问自己，“如何重组过程才能快速发现变化，更容易实现变更？”

为价值要求建立流程图

价值流图是价值创建工作的端到端流程图，它的目的是帮助你理解价值要求的工作流，这样你就能改进流程。通常价值流图从客户有一项需求开始，终止于需求得到满足。但是，上市时间图可能从产品概念得到批准开始，终止于客户开始从该产品中实现价值。在你开始画价值流图之前，利用一个时间序列图，让平均每项需求通过系统的端到端流程时间可视化。这就是你目前的能力。价值流图将告诉你有多少时间花在增加价值上。㊀

价值流图

Henrik Kniberg展示了一个视频游戏开发公司的开发过程的价值流图㊁。该公司目前的开发时间很长，这导致了错过市场时间窗和很高的开销成本，更不必说缺少能成功开发产品的人所带来的影响了。

Henrik这样描述价值流：

当某人（假定叫Sam）有一个新游戏的想法时，他花几个小时来准备

㊀ 更多价值流图的例子可以在Poppendieck,《Implementing Lean Software Development: From Concept to Cash》, 2006, 第83～92页中找到。

㊁ Crisp公司（一家斯德哥尔摩的公司）的Henrik Kniberg在斯德哥尔摩的Deep Lean Conference会议上画了这张价值流图。时间是2008年9月。Crisp公司的Mattias Skarin帮助我们理解这个例子的更多细节。这张图的使用得到了许可。

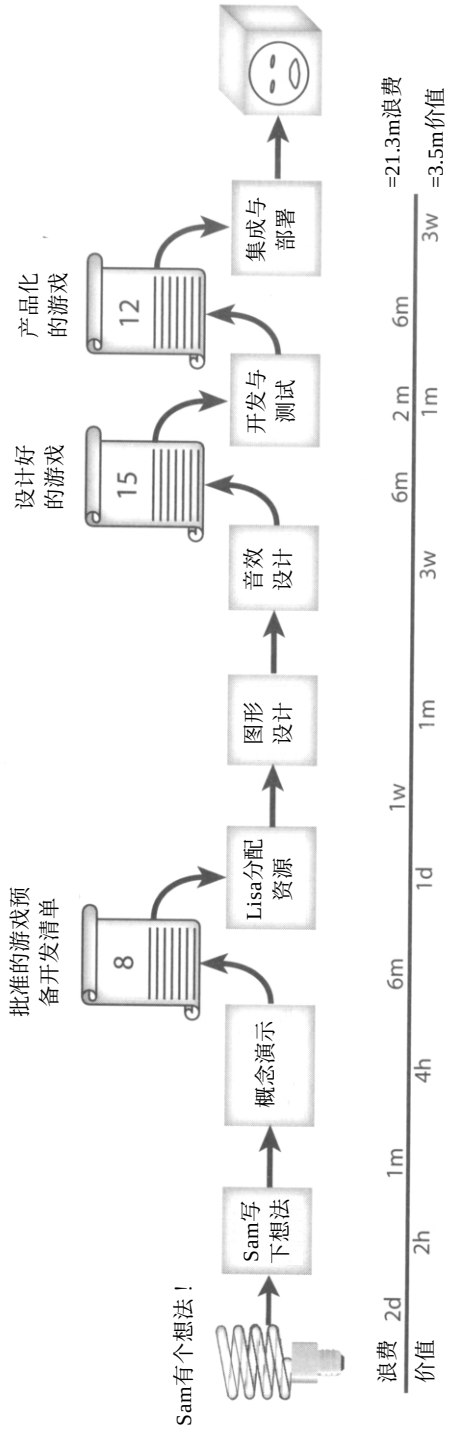

图1-4　上市时间价值流图

概念演示。大约在一个月后，这个演示到了想法复查委员会，因为这是一个好想法，它进入了游戏预备开发清单。但是队列中不止有一个游戏，还有其他一些游戏，所以在6个月之后，图形和音效设计师才会有空（见图1-4）。

当设计师有空时，Lisa就给他们分配一个新游戏，两个月后设计完成了。在这两个月中，只浪费了一周时间，但当游戏设计好之后，它放到了已设计队列中，在这个队列里苦苦等待了6个月。当开发团队最终拿到这个游戏时，同时还在开发另外两个游戏，所以它花了三个月来完成开发，然而团队实际上只花了一个月的时间来开发Sam的游戏。当团队完成开发后，游戏在产品化的队列里又等了6个月，等待最后的集成和部署。最后，在Sam提出思想之后两年，这款游戏发布了。在这两年中，只有三个半月（或14%的时间）是用于为游戏增加价值的。

诊断：

显然，在这个系统中存在太多过程中的仓库，太多的工作传递，以及许多的多任务。为了同时处理所有这些问题，公司成立了一个跨职责的团队，每个团队一次开发一个游戏，以端到端的方式，没有工作传递。每一个团队表明，它可以在少于4个月的时间内重复开发新游戏（比以前快6倍）。

Mary和Tom Poppendieck

寻找最大的机会

价值流图的目标是帮助定位机会，来改进处理能力。当你比较增加价值的时间和总体处理时间时，你会感觉到事情本来可以变得多好。问一问你自己，“为什么我们有这么多不增加价值的时间？真的有必要吗？原因是什么？”一般价值流图中最大的延迟或循环为改进过程能力提供了最大的机会。我们建议你找出最大的机会并利用它。在改进过程时忘掉价值流图，在你改变了过程之后可以画一张新的价值流图。

不要简单地理解价值增加时间与总处理循环时间的比率（处理过程的效率）。我们发现，在处理过程改进后，新的价值流图可能表明处理过程的效率下降了。为什么？因为在画第一张价值流图时，很多的浪费没有注意到，当

你习惯了发现浪费和消除浪费，你才会找到更多的浪费。

取景框4：政策造成浪费

浪费就是耗尽时间资源、工作资源、位置资源和金钱资源，又不为客户带来价值的东西。绝大多数人没有从浪费的角度来看系统，所以浪费常常在他们的视线之外。当你看到浪费时，就可以消除浪费，所以重要的是改变你看待工作的方式，从而让一直存在的浪费变得明显可见。

你可能会惊奇地发现系统中存在的部分浪费是由系统本身所引起的，或是系统完成工作的方式所造成的。许多情况下，浪费隐藏在习惯或传统智慧的外衣之下。通常，这些浪费的资源嵌入在组织机构的政策和标准流程中。除非这些政策改变，这些浪费是不会消除的。

政策怎么会造成浪费

政策造成浪费的方式有许多种。例如，许多公司的领导者相信开发者不应该与客户交互，因为这浪费了宝贵的开发者时间。我们前面讨论了关键缺陷处理过程，这个过程中有3级客户支持将开发者和客户隔离开来（参见图1-3）。通过简单地消除这3层客户支持，让开发者直接与客户交谈，节约了800名开发者40%的工作时间。这不是一个孤立的案例，我们还会在第6章的案例分析中看到，开发者与客户的直接互动可以交付更多正确的东西，增加销售，大幅减少支持电话。

当然，简单地提供客户与开发者的直接交互不一定能消除最大的浪费，某些政策还可能破坏良好的客户交互所带来的好处。

“我们觉得自己像是游戏中的棋子”

“我曾在一个开发团队中，在客户看来，我们有很强的拉动刺激。如果需要，我们可以在2周时间内添加特征并调整开发优先级。我们实际上非常努力，让一个瀑布式的公司变得更灵活，并且在某种程度上取得了成功。我们聆听客户的意见，真正使用产品的客户。我们很快地创建发布版本。

“我们开发团队没有意识到，产品安装有多么困难。当我去现场使用

产品进行升级时，我渐渐理解了这一点。最终用户问我，为什么他不能够自己升级，并指出他是一个有经验的老用户，对计算机和编程有许许多多的经验。我告诉他我不怀疑他的能力，并解释对我来说事情是怎样的。我被告知要飞行1万公里，穿过欧洲来升级系统，这个系统本来可以设计得更易于升级，如果有人早一点想到的话。我不断告诉他我多么希望事情应该是那样，我们越来越感觉自己像游戏中的棋子。我猜他更是这样想的。

“后来我意识到：升级和对应用生成的数据负责是合同上写明的。客户不允许升级，这是受合同控制的。”

“所以，我想，不论你在开发产品时的意图是什么，都很容易被早期的想法（或缺少想法）所摧毁，即如何对待你的客户。”

Ola Ellnestam瑞典Agical AB公司CEO、总裁和敏捷培训师

另一个例子也说明了政策造成的浪费，就是前面小节中上市时间价值流图（参见图1-4）。快速扫一眼就能发现3个很长的等待队列是导致产品延迟上市的主要原因，但是该组织却对它们视而不见。这可能是因为队列不归任何部门经理负责，真正的目的是为了缓冲相邻部门的绩效涨落。未能看到队列的影响通常是因为只关注部门级的绩效，试图充分利用稀缺的天才。这些政策导致大多数公司忘记了对上市时间和端到端的工作流进行大量拉动优化，从而影响到成本、收入，甚至也影响到资源的使用。

五项最大的政策浪费

根据我们的经验，在软件开发中，政策导致的浪费最常见的原因是：

1）复杂性

2）规模经济

3）将决定和工作分离

4）一厢情愿式思考

5）技术债

复杂性

在《No Silver Bullet》一文中，Fred Brooks写道：“相比其他人类创造的

东西来说，软件实体相对于它们的规模来说更复杂……开发软件产品的许多经典问题都来自于这种基本复杂性，以及复杂性随着规模的非线性增长。”我们都知道这一点。[⊖]但是仍然……

1）我们的软件系统包含了太多的特征，这些特征从没被使用过。[⊖]这些多余的特征还增加了代码的复杂度，使成本非线性增长。假设只有一半的代码是不必要的（这是保守的估计），系统在这些多余代码上的成本就不止是翻番，可能是所需成本的10倍。改进软件开发生产效率的最大机会就在这里：停止将并非绝对必需的特征放到了系统中。

2）我们的许多政策断定，范围是不能谈判的。这让我们无法停止加入这些没有必要的特征。我们需要一个过程，让我们能够开发系统的20%，让它投入使用，听取反馈意见，然后在时间和金钱允许的情况下添加特征。我们需要这样的政策：如果某方面需要折衷（成本、进度或范围），那么缺省的选择绝对是范围。

3）甚至我们的测量指标也隐隐传递出这样的信息，即我们应该往系统中添加尽可能多的代码。我们通过代码行数或功能点数来测量生产效率，好像这些就是好东西。它们不是好东西，它们不好。功能点可能提供有趣的相对数据，但永远不应该成为绩效测量指标。

4）我们需要保持代码集的简单性。这意味着我们应该在需要的时候再添加特征。忘掉有可能的需要，要用时即时开发。我们需要形成增量式开发的架构。我们需要政策保证重构（在改动代码时消除复杂性）成为添加新特征时正常而期望的步骤。

5）我们的客户常常希望用软件来自动化他们的复杂流程。这不是个好主意。业务过程应该先简化，再自动化。有多少时候，我们是先帮助客户简化流程，再进行流程自动化的呢？

精益参考取景框关注简单性。精益思考者知道，复杂性会阻碍工作的流动，让事情不可避免地慢下来。复杂性的成本是隐性的，它对成本有着二阶效应，所以我们在财务系统中看不到它。这让复杂性害处更大——很难在成

⊖ Brooks, “No Silver Bullet: Essence and Accidents of Software Engineering,” 1986。

⊖ 更多的细节请参见Poppendieck, Implementing Lean Software Development: From Concept to Cash, 2006, 第24页。

本上说明为保持简单而花钱是值得的。

在本章的末尾我们引入了构思（ideation）的概念，即得到一个设计来处理似乎不可避免的问题的过程。解决方案要做到“刚好合适”，就一定是简单的。了不起的设计总是让我们诧异，为什么这么明显的东西却被我们忽略了这么久？

缩减范围，满足最后期限

“我们做得很好，”他们说，“代码集很健壮，我们的速度很稳定，我们对系统实现的方式感到自豪。但我们的速度还达不到管理层的要求。所以有一天，老板跳过高级人员，向我们许诺了一个非常大的红包，如果我们能赶上最后期限的话。我是说，那个红包真大！”

我对他们的坦率感到吃惊，因为他们是在公开的会议上演讲。

“我们同意为之努力。我们日夜加班，包括周末。我们尽可能快地垒起代码，并停止了测试，只要能工作，测试不重要，测试只是必须进行。我们做到了，并得到了红包。”

“但是代码一塌糊涂。没有什么能正常工作。绕过的办法很糟糕，我们需要花几年来清理它。”

我并不吃惊。我可以断定他们不会为他们的所作所为感到自豪。“你认为你们本应该怎么做？”一名听众提问道。

他们很快回答：“我们应该说不，应该慢一点。”

“我没这么肯定。我一年以前去过他们公司，我尊敬他们的老板。他是一个好的经理，在一个不可能的职位上，我猜是这样的。我敢肯定错过最后期限是不可取的。”

后来在休息时我问两位发言人，“有可能缩减范围赶上最后期限，同时又不放弃测试原则吗？”

“哦，是的，绝对可以！”他们马上回答道。“我们写的代码有很多还没有用到过。但是我们不能。当你一年前来我们公司的时候，你告诉我们的管理层，如果他们想成功，可以缩减范围。所以一名高级工程师走遍了所有的经理，拿到了将一部分功能推迟到下一个版本的许可。但在那以后，没有再愿意缩减了，我们甚至都不愿意提起。”

我想管理层离工作不够近，意识不到延迟实现一些主要特征是容易而符合逻辑的选择。我想他们不知道设置不可能的目标，再让团队去想办法实现目标所引发的灾难。这是个代价巨大的错误。

根据我的经验，缩减范围并赶上最后期限几乎总是最好的方法。

Mary Poppendieck

规模经济

我们的许多直觉、政策和过程都植根于规模经济，它在20世纪上半叶的工业化生产取代手工生产的过程中，带来了巨大的生产效率提高。但在20世纪的下半叶，人们明显注意到，在高度多样性的系统中，流程的经济性要超过规模的经济性，即使在制造业中也是如此。软件团队开发独一无二的系统 —— 这是多样化精髓。很明显，我们应该将政策和过程建立在流程经济性的基础上。但是仍然……

1）很难放弃批处理和排队的意识。我们把工作形成批次，这样就能为每个批次指派合适的专家，最大化地利用专家的时间和技能。充分利用最有技能的员工是十分重要的，个人的环境让他们关注于自己那部分工作，不考虑对下一步工作或最终客户的影响。结果，我们的工作流和工人都不能处理多样性。

2）很难放弃批处理和排队的意识，当队伍就在我们面前时我们都看不见。我们不明白为什么需要花这么长的时间让需求通过我们充满未交付订货的过程。我们看不见客户请求的列表可能需要几年才能清空。我们不能让队列变短，因为这可能意味着对客户说不，而不是让他们的请求慢慢死掉。

3）我们不是设计一个系统来吸收紧急的请求，而是让工人放下他们现在的工作，投入到通过系统的更重要的工作。我们要求人们同时处理三、五、十项或更多项任务，并没有看见大量的时间浪费在上下文切换中。我们从未注意到，如果我们一次做一件事，而不是三件事，所有事情的完成就会快得多，从而会更快地交付价值。

4）有时候我们确实让人们一次处理一件事，然后用计算机系统来确保每个人所有时间都在忙。我们为项目制定进度计划并指定人员时，考虑的是充分利用。这种进度计划不能够吸收一直存在的涨落，所以它在新组建的团队

的等待时间里是可以吸收的。为已经建立好团队指派工作，比围绕项目重新组建团队要好得多。

5）我们创建年度预算或长期项目计划，让每个人都承诺要交付什么。然后我们一次性地将一大堆工作交给我们的组织机构。我们必须交付所有承诺的东西，但随着时间推移，事情发生了变化。客户想要其他东西，但不想付更多钱，或放弃承诺的东西。我们知道这个系统行不通，但我们找不到其他办法来避免这种大批量承诺的政策。

精益思考利用流程经济性来为我们观察世界提供取景框，而不是规模经济性。多样性是软件开发的基本特点，所以我们需要一些过程能够很好地吸收多样性。我们将在第3章讨论这样的一些流程。问题是，当世界已经被规模经济性框住时，这些方法似乎有些违反直觉。

将决定和工作分离

任何问题在离开它的源头时都需要简化和抽象，正因为如此，设计不应该脱离实现它的具体上下文。我们知道，对于工作的最深刻的见解是基于我们在现场获得的隐式知识：观察、体验，让我们的手变脏。我们知道了不起的设计来自于沉浸在解决问题中的设计者。我们知道把工作抛过墙头是行不通的。但是仍然……

1）很多人相信，经理不一定要懂得他所管理的工作。但是如果没有技术背景，经理就不能够为技术工人提供指导。有些经理只是简单地建立目标，让工人们去想办法如何实现目标。另一些经理组建起团队，授权让团队成员想办法去完成正确的事。从精益的观点来看，经理的基本工作就是理解他们管理的工作是怎样工作的，然后关注于如何让工作变得更好。[⊖]这并不是说所有领导者必须知道所有答案，关键是他们知道要问什么问题。

2）我们创造的组织文化，唯一的职业发展途径就是抛开技术细节。我们不尊敬，也不经常奖励有经验的架构师或出色的用户交互设计师。对于一个能够重新设计测试过程，并找来一组工具，每天晚上在各个操作系统和数据库上执行每一行代码的技术大牛来说，等待他们的未来是什么呢？如果这些

⊖ 参见Seddon,《Freedom from Command and Control: Rethinking Management for Lean Service》, 2005, 第4章。

人唯一的职业发展途径就是离开他们的核心专业知识并成为经理，我们就永远不能得到最优秀的技术领导者，而我们需要这种领导。

3）在《Lean Product and Process Development》一书中，Allen Ward说过脱手（工作传递）是产品开发中最大的浪费。他说，当我们分离职责（要做什么）、知识（怎么做）、动作（具体做）和反馈（从结果中学习）时，就会发生工作传递。⊖我们的过程中充斥着这样的工作传递，我们看不出这样的工作传递有什么错。但在实践中，许多日常决定是基于隐式知识的，这些知识会在工作传递过程中丧失。如果我们开始理解隐式知识的话，就必须认为隐式知识是通过魔法传递的。

4）当我们谈论“那项业务”时，我们的语言背叛了我们。通过这些词语，我们将开发决定与它们自动化的工作分离开来（见图1-5）。

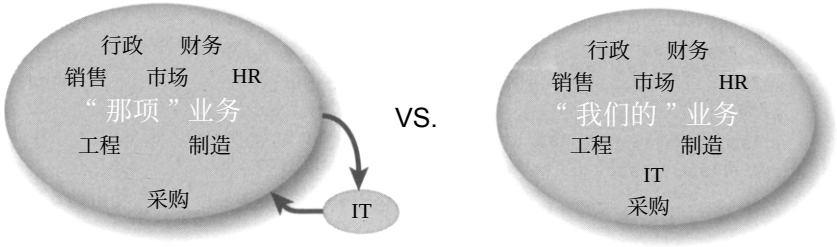

图1-5　从“那项”业务到“我们的”业务

甚至一些敏捷开发方法学也犯了这个错误。他们推荐一名“客户”或“产品拥有者”，假定此人可以决定所有客户想要的东西，并为开发制定优先级。但最成功的开发需要让开发者与客户直接交谈，并成为业务团队的一部分。那些所谓的需求是什么？它们实际上是候选解决方案，将需求与实现分离是另一种形式的工作传递。

5）在我们的系统部署之后，有政策和法律解释（如萨班斯·奥克斯利法案，Sarbanes-Oxley Act）要求开发者远离他们的代码。所以我们离开了，让支持团队来处理所有发生的问题。支持团队成员是那些在半夜里接听电话的人，而我们却在想他们为什么不喜欢频繁部署。他们是我们有风险的设计实践的受害者，但我们没兴趣听系统失效的原因。如果所有开发者每年必须抽出一个月时间加入运营和支持团队，也许我们的世界会变得更好。

⊖ Ward,《Lean Product and Process Development》, 2007, 第43页。我们使用工作传递（handover）而不是脱手（handoff），因为这似乎能够更好地翻译成别的语言。

自从亚当斯密写下了制针工厂的劳动分工后⊖，人们普遍认为劳动分工是一种增加生产效率的聪明思想——越专业化越好。幸运的是，这个“事实”在丰田的大野耐一（Taiichi Ohno）那里失效了，他设计了一个系统，让拥有多种技能的工人和容易重新配置的机器比专家更有生产效率。这样做有两个原因：首先，精益系统的设计目标是吸收多样性；其次，它们的设计目标是无情地改进，因为工人会设计出更好的工作方式。在系统开发的上下文环境中，亚当斯密绝对错了。

现场开发

什么样的CIO不喜欢读这类文章？

Standard Life的敏捷开发与SOA⊜

2007年10月18日 星期四

Investors Chronicle杂志不是以夸张著称的，但它肯定喜欢他在Standard Life看到东西……它的增长一直“有点缺少速度”，这本杂志感情冲动地说。半年的财务报表显示效率的大幅提高导致了营业利润提高71%，新业务增长31%。

因为这种镀金的苏醒，大量的信贷放在了Standard Life的IT部门门口……它创建了面向服务架构的最先进的实现，实践了精益过程，从根本上动摇了核心应用的开发结构，IT人员“嵌入”到了他们服务的业务部门之中。

这让我们印象深刻，因为我们两年之前去过那里。我们继续读下去：

CIO Keith Jones说，“对我来说，灯泡点亮的时候就是Mary Poppendieck说，大约60%的交付代码从来就没有执行过。我想这太让人吃惊了，肯定有一些可怕的公司做了真正糟糕的事情。然后她说，‘顺便说一句，我们检查了Standard Life公司的一个例子项目，你们的数据大概是64%’。”

这篇文章继续说，领导层问自己，为什么让分析师、开发人员和他们的客户坐在不同的地方。为什么不让开发软件的人与业务团队坐在一起，

⊖ Smith,《An Inquiry into the Nature and Causes of the Wealth of Nations》, 1776。

⊜ Swabey, “Agility Applied at Standard Life,” 2007。

聆听客户的痛苦，与创造业务用例的人交谈呢？所以Standard Life创建了一个部门，大致是一半信息系统员工和一半退休金部门员工，他们都坐在一起，组成一个团队。

将开发者放到业务团队中是一种成功的模式。我们曾在一家银行开过一个班，他们扫描抵押贷款结清的相关文件，进行数字化存档。它为文件扫描过程进行了大量的精益工作，问题是精益团队找不到人来改变控制他们工作的工作流软件。这种变更似乎很小，所以它们不能获得足够的优先级。精益团队的领导者抱怨说，“在过去的18个月中，我们提交了1536份请求，要求修改这个软件，没有一项请求得到实现。”

我知道他们正在使用的工作流软件，而且我也很肯定它是相对比较容易配置的。所以我建议他们将一些开发者暂时放到运营团队中，让他们帮助团队成员走出困境。后来我们听说这种方法非常成功，很多软件开发工作现在都已经由开发者完成了，他们嵌入在跨职能的业务团队中。

Mary Poppendieck

一厢情愿式思考[⊖]

Frederick Winslow Taylor弄对了一件事情。他坚持工作改进应该基于科学的方法。大野耐一拥抱了这个思想——但是他没有让“专家”来测量和改进产品工人的工作，而是培训产品工人来测量和改进他们自己的工作。我们知道基于数据而非观点来做决定是正确的方式。在软件行业，我们也可以创建一些工具来收集任何一天想要的数据。但是仍然……

1）我们追逐软件开发的最新思想，但不关心科学的方法。我们认为理解理论、创建假说、进行试验、收集数据、发现在我们的环境中真正有效的工作方法是浪费时间。我们没有认识到“最佳实践”是别人对他们的问题的解决方案，不一定适用于我们的问题。我们一厢情愿地采用新的开发方法，而不是确定我们的环境下最适合的实践：然后我们对失望的结果很吃惊。

2）我们在管理时查看的是单点的数据，而不是在上下文中的一系列数据。我们设置目标时不理解过程的能力与目标的关系，从而不能够认识到这

⊖ 将一厢情愿式思考看成是浪费，以及本节中的许多思想，都来自于Allen Ward (《Lean Product and Process Development》, 2007)。

一事实，即试图消除一般（常见原因导致的）涨落将使情况变得更糟，而不是更好。

3）我们不喜欢不确定性，所以我们试图做出决定消除不确定性。我们的天性偏好寻找另一种解决问题的方法，因为我们认为这比多看几种可选方法更便宜、更快。对于麻烦的问题，通常这都是错的。当我们都还很无知的时候就早早决定，这是最不可能得到好结果的，很可能是迫使我们重新开始。

4）尽管我们在处理信息方面技术非凡，但保存知识的技术却很少。我们的方法一直是收集大量的、详细的文档。但是谁会去读这些文档？甚至搜索引擎都不能满足我们的要求。另一方法是白板，带有一个照相机，如果我们确实需要保存草图的话可以用它。 还有另一种方法是摄像机，将白板讨论应用基础架构的过程拍下来。当然还有一些结合的方法，但我们仍在寻找。我们也许可以从开源运动中学到一课，即所有的沟通都是书面的，重点是让沟通系统非常简单、比较简明、容易查找，永远不要用口头沟通来代替。

5）当代码通过测试时，我们会有很大的成就感，会举行庆祝会，因为它满足了需求规格说明书 —— 好像规格说明书可以包含我们需要思考的所有问题。那些安全漏洞、内存泄漏或从数据库的异常退出而偶尔导致的死锁怎么办？系统的安装、操作和填充数据是否容易？当回归测试完成时就认为我们完事了，这是一厢情愿式思考。

如果你在发现计划的缺点时想到了学习，计划驱动的开发就黯然失色了。相反，创建有用的知识成为开发新产品的精髓。但是我们要学习的不止是在开发的产品，也要学习开发产品的过程。不断学习是改进产品本身和产品开发过程的精髓。

技术债

我们知道所有成功的软件都会修改。[⊖]所以如果我们觉得手上的代码将取得成功，那就需要让它易于修改。

所有让代码难以修改的东西都是技术债。[⊜]我们知道技术债无情地推高了软件总体拥有成本，最终我们将不得不为之付出代价，系统最终会破产。

⊖ 参见Brooks, “No Silver Bullet: Essence and Accidents of Software Engineering,” 1986。

⊜ 技术债的隐喻是Ward Cunningham引入的, “The WyCash Portfolio Management System,” 1992。

但是仍然……

1）我们容忍模糊的代码，而不是确保所有的代码都反映了它的意图，让后来的人能够清楚知道。开发者，特别是新手，应该学习如何编写“干净的代码”㊀：简单、直接、逻辑直白的代码。高级技术人员需要确保混乱的代码即使能够通过测试，也不允许其进入代码集。

2）我们通常不花时间进行重构：对已有的代码进行加固修改。重构对于迭代式开发是十分重要的。为已有的代码添加新的特征将带来复杂性、二义性和重复，重构就是还这些债。

3）我们在部署系统之前进行回归测试。开始它们很快，但随着每次添加代码，回归测试变得越来越慢。当回归赤字（regression deficit㊁）增长时，我们发布版本的间隔时间变长了。要打破这种增加发布开销的无穷循环，唯一的方法就是减少回归赤字。如果我们在代码还少的时候就有自动化的测试套件，并不断添加和维护测试套件，就可以在今天以几乎同样快的速度，修改新的代码集。

4）我们知道依赖关系是技术债最大的来源之一，但是我们却矛盾地用大量的依赖关系来代替过时的系统。我们必须开发依赖关系最小的架构，并迁移到这种架构上。我们很久以前就知道怎么做了：关注信息隐藏㊂和关注点分离㊃。

5）我们因为许多原因对代码进行分支：为了隔离新的开发，为了关注单个应用，为了创建平行的特征集。我们知道两个分支的代码分开的时间越长，它们就越难合并在一起。但是我们仍然要等上几天来构建代码，更糟糕的是，系统测试还被推迟到开发结束的时候进行。我们没有意识到事情已经不再是这样，大爆炸式的开发已经过时了。

我们需要揭露技术债的本质：它是代价高昂的负担，要避免它，以免导致我们破产。第2章将透过技术的取景框来看软件开发，讨论避免技术债的一些坚实的技术。

㊀ 参见Martin,《Clean Code: A Handbook of Agile Software Craftsmanship》, 2009。参见第7页的定义。

㊁ 我们第一次听说“回归赤字”这个术语是在Owen Rogers那里。

㊂ 信息隐藏意味着将一同改变的特征放在同一个模块中。参见Parnas, “On the Criteria to Be Used in Decomposing Systems into Modules,” 1972。

㊃ 关注点分离意味着将独立变化的特征放在不同的模块中。参见Dijkstra, “On the Role of Scientific Thought,” 1982。

肖像：产品捍卫者 —— 理解1

我们在本章结束时来看看系统开发过程中可能是最重要的部分 —— 虽然绝大多数人认为它可能发生在开发过程开始之前。在你一头扎进开发系统的工作之前，重要的是从客户的角度来定义目标，就像本章前面从客户的角度来定义组织机构的目标那样。

当爱迪生在1879年发明电灯泡时，它是个相当没用的新鲜事物。所以他发明了电力传输系统，建立了一个公司来传送电力，建立了一个蒸汽发电厂 —— 为了让电灯泡得到广泛使用。爱迪生的天分就在于他能够预见人们想要使用电灯泡，并想象一个充分开发的市场。我们在Steve Jobs身上看到了同样的天分，iPod和iPhone进入了人们的生活，带有完整的生态系统。这是对开发的基本挑战：想象人们会怎样想用一个产品，预见一个完整的系统和一个充分开发的市场。我们称之为构思（ideation）㊀。

正如爱迪生去创建公司，从而让他的电灯泡大量生产那样，我们期望有构思能力的领导者在他们的概念实现时保持领导位置。毕竟，人们对他们的创造性想法产生了一定的热情，渴望将这些想法带入生活。为了向这种热情和奉献表示敬意，我们将这样的领导者称为产品捍卫者。㊁

产品捍卫者很像一名企业家，对产品的成功和失败负有业务上的责任。这意味着对于一个有损益表（P&L）的产品来说，他们对产品的P&L负责。这就是为什么产品捍卫者领导构思工作的原因 —— 如果构思不好，产品便不会取得成功。㊂

首席工程师的故事

Nobuaki Katayama先生曾是Lexus/SC、IS和Altezza的首席工程师，在2009年4月的一次日本学习访问时，与我们讨论了首席工程师的工

㊀ “构思”这一术语来自Tim Brown, IDEO的CEO, 出现在“Design Thinking,” 2008中。

㊁ “产品捍卫者”这一术语来自3M公司，Mary在那里工作了20年。同样的角色也常被称为“首席工程师”，可以和“产品捍卫者”互换使用。

㊂ 参见Levine,《A Tale of Two Systems: Lean and Agile Software Development for Business Leaders》, 2009,该书给出了软件开发中首席工程师的一个深入的例子。

作。⊖他指出，新车开发是由首席工程师领导的，他负责这款车的商业成功。此人应该对这款车很有热情，所以说如果是辆跑车，首席工程师就应该喜欢飙车。

丰田的新车开发分为三个阶段：创造、开发和生产

创造：新车的创造阶段就像开发过程的支柱，也是最难的部分。在这个阶段进行计划和概念开发，包括市场调研、预开发、定型、成本和利润目标等。首席工程师与副手们（汽车外形、引擎、传动等部分的负责人）讨论，研究这款车要具备怎样的性能和关键特征，开发这款车需做什么事等问题。创造阶段没有时间限制，工作一直持续到概念完成，这通常需要一年的时间。人们达成了握手协议，产品概念得到优化并进行复查，最后由董事会批准。

开发：这是创造新车最容易的部分，丰田擅长这项工作。首席工程师建立起一份20到24个月的进度计划，并领导此项工作，由各部门经理提供支持。有一些清晰的里程碑需要实现，当然，在这个过程中还有更多的讨论和不断的设计决定。

要保持开发按进度进行，就要诚实的沟通和“先说坏消息”的哲学，这样问题就能尽快得到解决。开发过程中的关键决定是从一些充分讨论的可选方案中选择的，这样就能够实现最佳的品质、成本和交付组合。

生产：在这款车的生产和整个生命周期中，首席工程师仍然负责它的商业成功。

Mary和Tom Poppendieck

为了简单，我们使用“产品捍卫者”这个术语，但是我们意识到你可能不会使用“产品”这个词来指你的系统。你可能在开发：

1）作为产品的软件

2）嵌入在产品中的软件

3）支持一个过程的软件

⊖ 使用这次会面的小结得到了允许。故事中的某些信息来自Kenji Hiranabe在Agile 2008大会上的演讲和Nobuaki Katayama先生在Developers Summit 2008大会上的演讲，这次会议于2008年2月13日,在东京的Gajoen召开(Hiranabe, “New Car Development at Toyota,” 2008)，使用得到了允许。

4）合同规定的软件

在前两种情况下，产品捍卫者应该是负责最终产品的业务的人。对于较大的系统，产品捍卫者可能需要助手来领导子系统的开发。即便如此，切分子系统也不应该沿着技术路线进行，而应该沿着子系统生产线，例如，汽车的引擎、医疗设备的编程器、电子设备的人机界面等。

在第3种情况（支持一个过程的软件）下，特别重要的是产品捍卫者要负责总体过程的设计和成功，而不只是软件（实际上在这种情况下，他是过程捍卫者）。

第4种情况（合同规定的软件开发）对产品捍卫者来说是问题最大的，特别是如果合同签订方将软件开发与系统其他部分的开发分割开来的时候。最后，签合同的软件开发者承担了范围更宽的责任，实际上最好是有一个产品捍卫者来承担这种责任，指导系统开发过程中的学习和必要的反馈。我们意识到这并非总是能够做到，但不管在什么情况下，产品捍卫者必须在脑子里思考整个系统。

产品捍卫者承担两项关键职责：面向客户的构思角色和面向技术的构思角色。通常这些角色由一个人来承担，但也可以由相互协作的两个人来分担。不管是哪种情况，产品捍卫者通过领导团队进行构思来启动开发。

面向客户的构思

IDEO是加州的一家设计公司，在发现未满足的需求并匹配到可行的技术和可行的商业模式方面，他们做得特别成功。这导致了一个惊人的产品线，让客户特别高兴。当年复一年，他们的设计奖项堆起来的时候，IDEO开始进入一个新的业务领域，帮助其他公司复制它的设计过程。㊀

总经理Tom Kelley 在《The Art of Innovation》一书中概括了IDEO的设计方法㊁：

1）理解市场、客户、技术，和观察到的问题约束条件。稍后我们会常常挑战这些约束条件，但重要的是要理解当前的感觉。

㊀ 这一部分的内容来自Brown的总结集, “Design Thinking,” 2008。也可以参见Brown, “Strategy by Design,” 2007。

㊁ 摘自Kelley和Littman, 《The Art of Innovation》, 2001, 第6～7页。

2）观察真实的人的真实生活状况，发现什么使他们难过，什么让他们迷惑，他们喜欢什么，他们恨什么，他们有哪些潜在需求是目前的产品和服务所不能满足的。

3）想象未来世界的概念产品和使用它们的客户。有些人认为这一步是预测未来，而且它可能是整个过程中头脑风暴最集中的阶段。

4）用一系列的迭代来评估和优化原型。我们试图不要太深入前几个原型，因为我们知道它们会被挑战。没有什么想法是好到不需要再改进的，我们计划了一系列的改进……我们观察什么有效、什么无效、什么让人们迷惑、什么让他们喜欢，我们增量式地改进产品。

5）为商业化而实现新的概念。

我们找不到更好的关于构思的总结了。先框定问题的约束条件，然后再成为一个族群研究者，在这个取景框下观察人们。这不是关于特定的人群，也不是市场调研，而是出去观察会使用该产品的人，并深入到他们的内心。

下一步就是设想、建模和讨论观察到的东西。混入对未来几年技术趋势的前瞻。你想滑到冰球将要去的地方，㊀而我们的技术冰球移动得很快。头脑风暴、编造场景、讲述关于客户的故事、制作一些原型，让思想保持活跃。

设想将导致评估，一系列的快速实验增量式地改进产品。“你有多聪明并不重要，关于某事的第一个想法总是不对的，”IDEO的CEO Tim Brown说，“所以做原型（快速而廉价的原型）的重要价值就是，你可以从这个思想中学习，让它变得更好。”㊁

原来他们是这么干的

我一直喜欢厨艺，许多年来，我厨房里的那些器具一直都是我从小看到大的那些东西。后来一家名为OXO的公司发明了一种新的量杯，被称为带角度的量杯，它有缩进，这样你就可以从杯子里面看到液体的平面。我立刻就爱上它了，不用再弯下腰或举起杯子，并从杯子外面来读数了，而只要看一眼杯子里面就可以读出数据。我买了几个自己用，又买了许多

㊀ 冰球明星Wayne Gretzky说出了他成功的秘密：“我滑到冰球将要去的位置，而不是它现在的位置。”

㊁ Brown, “The Deans of Design,” 2006。

作为礼物送人。

然后我开始注意其他OXO的产品：有一个剥皮器比我手上的好得多。器皿很容易存放，非常好的砧板，很快我的厨房里就满是OXO的产品了。我在想，“一个公司怎么能够经常对沿用了几十年的厨房用具进行改进呢？他们怎么知道我不愿意弯下腰来读数据呢？又怎么知道我喜欢剥皮器的手柄大一些呢？我自己都不知道！”

然后我读到了IDEO怎么进行族群研究，他们称之为“追根溯源”。他们走进像我这种人的家里，看我使用量杯，以及寻找我感觉有些不舒服的时候——弯下腰或把一满杯东西举高。他们注意到当我拿起剥皮器时握得很不舒服。然后他们发明了这些东西，消除我都没注意到的那些烦恼。

原来他们是这么干的！非常简单，真的。你会想，如果我们使用同样的方法，我们可能会得到类似的结果——得到一些超乎想象的系统，解决了客户都没注意到的问题。

Mary Poppendieck

在适当的时候，产品概念可以准备实现了。构思不应该是一个很长的、持续很久的事情，得到的概念应该是高层次的，为进一步的学习留下了很多空间。

面向技术的构思

对于软件开发来说，从客户的角度来想象一个产品还不够。因为没有足够有效的技术远见（我们称之为架构），你就还没有准备好继续向前。与创建面向客户的产品观点的步骤一样，形成架构愿景有如下步骤：

1）理解：技术永远在变化。从理解技术发展的现状开始，特别是要理解在产品的生命周期中，技术将发展到什么水平。

2）观察：花一些时间来观察人们与问题搏斗的情形。从技术的角度来看，面向技术的概念和面向客户的概念在系统中得到统一，它们是一起形成，相得益彰的。

3）想象：建模、讨论、头脑风暴、用穿刺测试来验证想法。⊖为这些

⊖ 穿刺测试是一个快速的技术原型，目的是测试技术方法的可行性。

讨论提供技术发展方向的准确信息，以及在整个产品生命周期中可能会发生什么。

4）评估：不要抓住头脑中的第一个想法不放，要做试验。选择3至4种可选方案，在实现时利用多种设计，再做出最后的选择。最重要的是要记住，好的软件架构不仅有利于短期个性代码的架构，也有利于架构的长期演进。

5）实现：在这个阶段，架构是一个技术愿景，将随着开发的深入和学习的过程不断成长和发展。现在是开始实现的时候了。

构思阶段在你觉得完成时就完成了，不应该设置最后期限。这并不是说构思会花很长的时间，通常不会这样。但如果没有创新的概念，就不会有创新的产品。所以不要忽略这个重要的步骤。要形成清晰的愿景，想象产品将如何满足市场需求，架构将如何支持这个愿景，开发将如何进行，我们把这些称为产品概念。产品概念不是一个详细的计划，它是一个继续开发的取景框。

详细计划与继续开发的取景框有什么区别？可以这样认为：计划会因学到新的知识而改变，而取景框提供了学习的空间。所以如果你在产品概念批准之后，又发现必须对产品概念进行实质上的修改时，那么它就太详细了，没有提供学习的空间。

你怎么知道构思完成了呢？这取决于不同的上下文环境，所以我们没有一个统一的答案。在拥有好的构思过程的公司中，产品捍卫者知道概念何时完成，并可以进入实现了。如果你不太肯定，就必须做实验，找出适合的东西。我们建议你利用行动偏好来开始试验。如果你还在想是否已经准备好实现，答案可能是肯定的。

练习

1. 回答John Seddon关于你的组织机构的五个问题：

1）目标：这个组织机构的目标是什么？

2）要求：客户要求的实质是什么？

3）能力：系统预期可以实现什么？

4）流程：工作是如何进行的？

5）系统条件：什么是系统中浪费的根源？

2. 为你的系统选择5个以客户为中心的测量指标，可以考虑以下一些指标：
   1）产品开发的上市时间（针对整个产品）
   2）客户请求的端到端响应时间（请求到解决方案的时间）
   3）产品在市场上获得的成功（利润、市场份额）
   4）新系统对业务改进的贡献（可以测量的业务改进）
   5）在交付后客户实现价值的时间（可消费性）
   6）漏掉的缺陷（发布后才发现的缺陷）的影响（客户宕机时间，财务上的影响）
3. 为你的以用户为中心的测量指标创建时间序列图。（有些人称这些图是“过程的声音”）如果数据还不存在，那么现在就是开始测量的好时候。
   1）这些图说明了什么？
   2）你的工作过程稳定吗？
   3）你可以区分常见原因导致的涨落和特殊原因导致的涨落吗？
   4）你的工作方式所交付的结果实现了用户预期吗？
   5）如果没有，应该做什么？
4. 分析你是如何处理来自客户的请求的：
   1）你每个月收到多少价值要求？
   2）你每个月收到多少失效要求？
   3）失效要求与价值要求的比例如何？
   4）你使用怎样的批准过程来过滤客户要求？
   5）什么评判标准？
   6）接受请求的百分比是多少？
   7）批准过程要花多少时间？
   8）客户平均需要花多少时间才能知道被拒绝的请求？
   9）你是否衡量事后的效果，看看提出要求的客户是否实现了他们的目的？
   10）批准的请求完成平均需要多少时间？
   11）客户对你的批准过程怎么看？
5. 召集一个团队，如果有负责端到端流程的人，请包含进来。画一张实际端到端流程的价值流草图，说明客户要求是怎样通过你的组织的各种处理过

程，再通过客户的处理过程，最后回到最初客户那里。

1）总循环时间是多少？

2）有多少时间用于增加价值？

6. 让团队进行头脑风暴，讨论组织机构中可能存在的目标、政策和信念导致的浪费。从下面五个方面进行讨论：

1）复杂性

2）规模经济性

3）将决定和工作分离

4）一厢情愿式思考

5）技术债

7. 你的公司中构思是如何发生的？

1）你的组织机构中是否有产品捍卫者或首席工程师这样的角色？

2）你们如何将思想从模糊的状态发展成为批准的产品概念？

3）效果怎样？