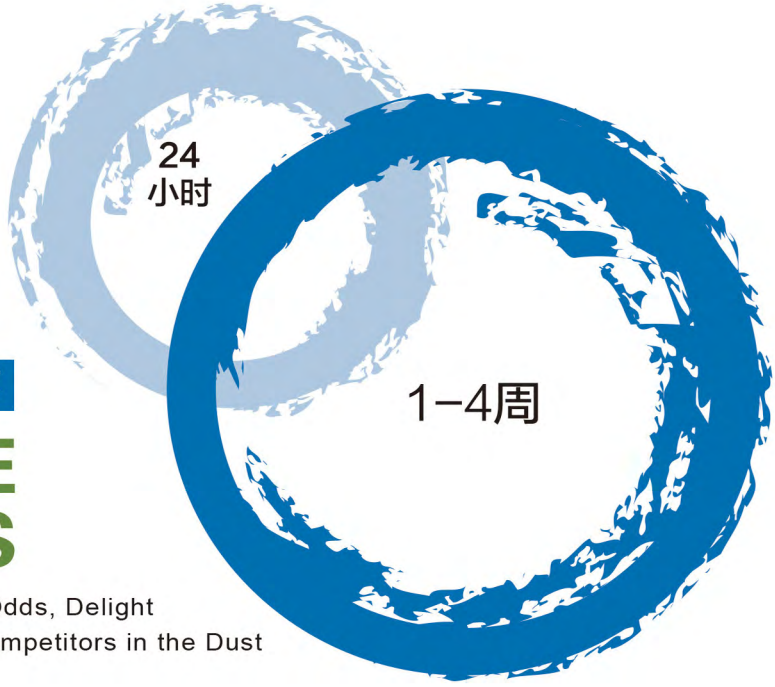
TURING 图灵程序设计丛书

30天软件开发

告别瀑布拥抱敏捷

[美] Ken Schwaber Jeff Sutherland 著
王军 李麟德 译



Scrum之父权威著作

SOFTWARE IN 30 DAYS

How Agile Managers Beat the Odds, Delight Their Customers, and Leave Competitors in the Dust

人民邮电出版社
POSTS & TELECOM PRESS

Ken Schwaber

软件开发专业人士，在过去40年的职业生涯中，曾担任过程序员、分析师、咨询师、产品经理，还做过企业家。过去20年里，一直致力于发展Scrum，并帮助世界各地的机构使用Scrum。他是“敏捷宣言”最早的签署人之一，也是敏捷联盟和Scrum联盟的创始人，目前正努力通过Scrum.org来改善整个软件行业。

Jeff Sutherland

马萨诸塞州剑桥市Scrum Inc.的首席执行官，专门为世界各地的公司提供培训、咨询和辅导服务。同时也是波士顿风险投资公司OpenView Venture Partners的高级顾问，帮助所投资的公司实施Scrum和敏捷实践。多年来，Jeff已在众多软件公司和信息技术机构推广和提升Scrum。

王军

全球领先的Scrum敏捷专业培训咨询机构ShineScrum的首席执行官，资深Scrum敏捷培训师和教练，CSP、CSM、CSPO，美国纽英伦中华资讯网路协会董事和中国区主席。有20多年海内外软件行业从业经验，曾任甲骨文上海BI研发中心总监，2000年在美国担任计算机集成制造（CIM）首席工程师。他辅导过国内许多大型软件企业导入敏捷，是引导个人成为优秀敏捷实践者的导师。毕业于北京理工大学，拥有美国百森商学院创业学MBA学位。

李麟德

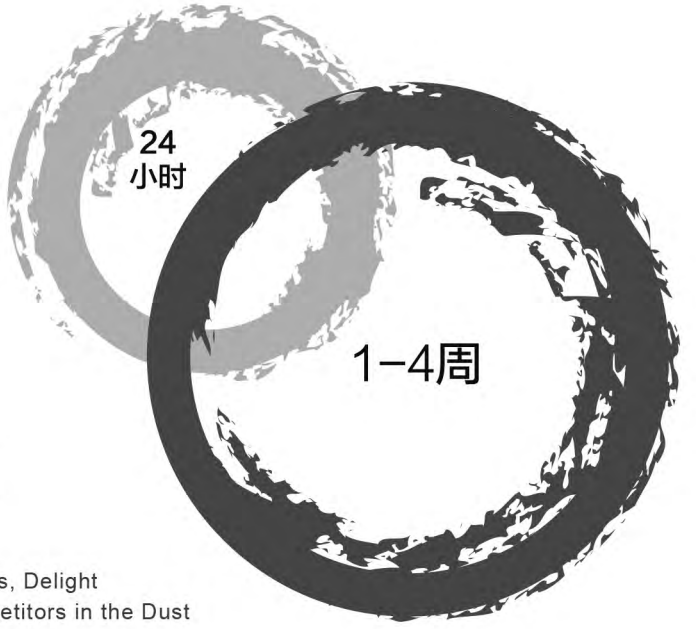
CSM、CSP。现任职于Oracle Endeca Information Discovery部门，专注于Java应用开发和自动化测试。拥有多年Scrum实践经验。

TURING 图灵程序设计丛书

30天软件开发

告别瀑布拥抱敏捷

[美] Ken Schwaber Jeff Sutherland 著
王军 李麟德 译



SOFTWARE IN 30 DAYS

How Agile Managers Beat the Odds, Delight Their Customers, and Leave Competitors in the Dust

人民邮电出版社
北 京

图书在版编目（CIP）数据

30天软件开发 : 告别瀑布拥抱敏捷 / (美) 施瓦布 (Schwaber,K.) , (美) 萨瑟兰 (Sutherland,J.) 著 ; 王军, 李麟德译. -- 北京 : 人民邮电出版社, 2014.1
（图灵程序设计丛书）
书名原文: Software in 30 days: how agile managers beat the odds, delight their customers, and leave competitors in the dust
ISBN 978-7-115-33889-1

Ⅰ. ①3… Ⅱ. ①施… ②萨… ③王… ④李… Ⅲ. ①软件开发 Ⅳ. ①TP311.52

中国版本图书馆CIP数据核字(2013)第293664号

内 容 提 要

本书讲解了 Scrum 敏捷软件开发方法，让你在 30 天内开发出全新的软件。读完本书，你会发现用敏捷开发方法能够让软件开发事半功倍，节省人力物力，大大提高工作效率。

本书适合于管理者、商务人士、小企业主、产品开发经理、IT 经理以及软件开发人员等阅读。

◆ 著 [美] Ken Schwaber Jeff Sutherland
译 王 军 李麟德
责任编辑 李 瑛
执行编辑 孙文潇
责任印制 焦志炜

◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路11号
邮编 100164 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京 印刷

◆ 开本：720×960 1/16
印张：11.5
字数：180千字 2014年 1 月第 1 版
印数：1－4 000册 2014年 1 月北京第 1 次印刷
著作权合同登记号 图字：01-2012-5776号

定价：39.00元

读者服务热线：(010)51095186转600 印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京崇工商广字第 0021 号

版 权 声 明

献　　词

谨以此书献给 Ikujiro Nonaka、Babatunde A. Ogunnaike 和 Hirotaka Takeuchi，感谢他们的鼓舞和指导。

中文版序

Scrum 好比打破瀑布式思维的敏捷破冰船船头。它用一种革命性的思维方式来思考工作和实施工作的人：软件开发是复杂的，其结果有时会令人诧异；工作是由人来完成的，人们的创造力和生产效率在小团队协作时最高。

我们都知道复杂的工作需要经常调整和修改。试想一下，仅仅是人们每天下班回家这件事就会做很多次调整。软件开发则要复杂得多，因为一切都是不可见的。思想能够建模，而软件最终只是一套组织严密的思想而已。

经验主义和精益思想是 Scrum 的两大基础。经验主义利用短周期循环，在每个循环结束时检视成果来控制复杂度，由此控制风险。短周期循环还能让精益思想发挥作用。所有浪费都变得可见并能够逐步被移除。

跨职能团队在短周期循环中履行责任、义务和潜在的实施时，其生产效率和创造力就会呈现出来。人们还能够发挥主观能动性，充分利用自下而上的智慧。

20 世纪 90 年代初，我和 Jeff Sutherland 在我们的公司中发起了 Scrum，并在 1995 年的 OOPSLA 大会上正式对外宣布。90 年代后期，我们对 Scrum 进行了验证和提炼，而后一直致力于它的传播和宣传。根据福瑞斯特研究公司（Forrester）2013 年的报告，全球已经有 92%的公司在使用 Scrum 了。Scrum 的使用已经从软件领域扩展到嵌入式产品、公关、数字媒体、竞选和剧院产业，现在还被企业高管用于对策略的频繁调整。

我们撰写此书就是要向世人宣布快速迭代的软件开发是真实存在的，而且就

在你的身边。任何人都可以让他们的软件开发人员在 30 天内开发出软件的一部分。他们不必煎熬于数月的悬念之中，反而能够在早期就知道自己的想法是否可行。本书提供了可能有帮助的案例、指引和提示。

需要谨记的是 Scrum 只是一个框架。Scrum 不会告诉你如何完成你的工作，而是提供一个地方，让你去实践并创造出尽可能好和尽可能有价值的产品。

Scrum 的核心是其价值观：做正确事情的勇气、竭尽所能的承诺、对当前问题的专注、对团队成员和相关干系人的坦诚，以及对自己和他人的尊重。承认并实现这些价值就能打造出优良的工作环境和出色的产品。将其扩展开来，则能创造出伟大的社区和社会。

祝你们的 Scrum 旅程一帆风顺！

Ken Schwaber
Scrum 创始人

推　荐　序

几天前，图灵邀请我为本书作序，希望我可以介绍盛安德这些年在敏捷实践方面的一些经验和教训。尽管心里有些忐忑，但我还是欣然应允了。

盛安德接触敏捷，缘于我们对欧美的软件外包服务业务。2005 年年初，一个英国客户希望能和我们建立一个联合团队，在伦敦、北京两地实施 Scrum。客户先派来一个四人团队到北京和我们的团队工作，两周后留一人常驻北京，并约定每三个月轮换一人。现在看来，客户是有备而来，在敏捷方面也已经积累了丰富的经验。

项目从 2005 年 9 月正式开始，有六位盛安德技术人员参与，大家都第一次听说敏捷开发。对一个只有五六十人的小公司，这个项目显得非常重要，作为公司负责人，我也投入不少时间关注项目进展。遗憾的是，这个项目在 2006 年 5 月以失败告终。但正是这个失败的项目，让我们重新审视自己的外包业务，对比一直在使用的瀑布式过程，Scrum 优势明显。Scrum 强调的透明、高效、协作原则，正是离岸外包业务最需要的特性。摆在眼前的问题是怎样在公司内部推广 Scrum 流程。

本书在组织导入敏捷的过程中，给出不少案例和方法，循序渐进是基本的原则之一，剧烈的变革给组织带来问题永远比效益更多。通过创建试点项目，组织的领导者可以感受到Scrum带来的高效。而领导者的参与和推动是组织实施Scrum过程的前提。人员培训是第一步，其目的是转变思想观念，对中国企业，这也是最大的挑战。再逐步深入，从单一的项目，到整个组织的敏捷化，逐步实现精益管理，最后让敏捷成为企业文化的组成部分。本书对 Scrum 实施的每一个步骤都做了详细的说明。

如果当时有这本书指导，或许可以帮我们少走许多弯路。无论如何，在 2006 年，我们开始在项目中尝试实施 Scrum。现在看，开始本身是一个困难的决定，好在一旦做出就不会回头了。尽管遇到许多意想不到的困难，但可以感受到团队和整个组织都在受益，困难也不再仅仅是传统管理者的责任，而是由大家共同面对。

2007 年开始，盛安德将敏捷理念融入离岸外包业务模型，培养和发展敏捷程序员也成为公司发展战略之一。随着 Scrum 流程开始在公司内部推广，我们发现 Scrum 很难在组织内作为一个特例单独存活，必须让整个组织融入 Scrum，实施自内而外的变革，才能让敏捷扎根于组织。

现在盛安德内部运营工作也采用了 Scrum 模式进行管理，每天早晨 10 点管理团队有 15 分钟站会（Daily Scrum），每周五下午进行回顾会（Retrospective，每月根据年度目标，由运营官作为 Product Owner 制定 Backlog，月底回顾。职能部门成员组成许多跨部门团队，采用相同的机制，解决日常管理过程中遇到的各类问题。如本书最后一章所述，我们发现 Scrum 可以成为组织解决管理和项目中各类复杂的问题的强力武器。

张纪伟

北京盛安德科技发展有限公司，创始人，董事长

译　者　序

2010 年 5 月我在波士顿拜访 Ken 的时候，他提及正在写一本专门给管理者的有关 Scrum 的书。这本书于 2012 年 5 月出版，我一口气就读完了，很快图灵教育给了我这个翻译机会。这本书是利用业余时间翻译的，前后断断续续一年多。英文原版不是很好理解，好多字句斟酌许久，尽量表达更符合中文的语意。

这本书是写给管理者的，是架起 Scrum 和管理者之间的一座桥梁，解释为什么要尝试敏捷。它是第一本能够提供完整的关于为什么要实施 Scrum 的栩栩如生的故事书。

本书共两部分，第一部分用大量的篇幅分析了瀑布流程出现的问题，并阐述迭代增量式的经验型软件开发方法能够解决复杂项目中的问题。敏捷思维贯穿全书。如果你想从本书学到更多的敏捷实践技巧，本书不太适合；但是，如果你是管理者，被软件开发碰到的问题纠结和困扰，想尝试其他办法，本书系统介绍敏捷思维，会打开你的思路。建议读者先从第一部分读起，看看书中阐述的敏捷思维的理念你是否认可。如果认同，你跨出了第一步。第二部分介绍如何渐进式地向 Scrum 转型及案例分享。附录的《Scrum 指南》是 2013 年的最新版，企业级敏捷攻略也是第一次公开出版。这里有几个章节我想着重介绍一下。

第 4 章给出了管理者在敏捷思维的驱动下，建立试点项目后，在 Scrum 实施中需要担任的角色。阐述经验主义是可能性的艺术，管理者需要给员工创造透明且安全的成长环境。经验型软件开发流程在组织里能有多成功，很大程度上取决于管理层在面对变革时的领导力和影响力。

第 7 章作者引入在组织内成立 Scrum 工作室的概念，这个软件工作室是组织内的一个全新、独立的机构。受访者从最佳实践和常识中对 Scrum 的认知调查表，是一个很好的评估工具。管理者都喜欢有管理指标的项目“仪表板”，用工作室中获得的数据提供了度量生产效率、质量和价值的有用指标。最后“技术债务”的罪恶之源示例及 Adode 的案例令人回味。

第 9 章介绍 Scrum 转型完成变革所需的一些活动和技术。第 10 章描述如何在企业里用 Scrum 的方式实施 Scrum，就是利用 Scrum 的流程来实现组织的转型。

如果你是一位管理者，没有时间阅读长篇的书籍，本书的简明扼要正合适你。书中每个章节都很精练，你也可以把书放在床头或办公室，随意挑选部分章节阅读，其中有大量的在整个组织内实施 Scrum 的典型案例真实分享。这也是我喜欢本书的原因之一。也建议你推荐这本书给你的上司和下属。敏捷转型是一场变革，如果没有高层的支持，项目组只能小打小闹，结果是要么停滞不前，要么中途夭折。它需要自上而下和自下而上的双向支持。

感谢图灵教育的信任，感谢朱新滨对本书部分章节的建议。也感谢 Ken 专门为本书中文版作序。

王军（Jim Wang）
ShineScrum
2013 年 11 月 4 日深夜
联系方式：jim.wang@shinescrum.com

前　言

我们俩在软件行业的打拼史加起来已有 70 个年头了。我们曾经在 IT 组织和软件产品公司担任过软件开发人员和经理，也曾创办过自己的产品公司和服务机构。20 多年前，我们发明了一套能够让组织更好地交付软件的流程。自那以后，我们用这套流程帮助了数以百计的组织。这项发明的广泛应用程度远远超出了我们的想象，现在已经有数百万人在使用它了，它就是 Scrum。Scrum 的广泛应用让我们深感谦微，大家使用 Scrum 所获得的成就让我们颇感惊叹。

关于软件开发我们之前写过一些书，不过为那些不亲身参与软件开发的人员写书还是第一次。本书专门写给其所在组织依赖软件保持生存和竞争力的领导者。他们的组织能够从快速增量式的软件开发中获益，从投资中获得最大回报，并需要解决由于业务和技术的复杂性使得软件交付困难的难题。我们写作本书的目的就是让这些领导者能够帮助他们的组织实现这些目标，增强其内在能力，改善其产品，等等。

本书是写给想用更短的时间、更低的成本、更好的可预测性以及更低的风险，交付更好软件的首席执行官、高管以及高级经理们的。我们想跟这些人说：在过去，你也许有过不愉快的软件开发经历，但是这个行业已经今非昔比，软件行业的策略和业绩已经有了彻底的改观。现在我们可以欣慰地说，你习以为常的不确定性、风险和浪费已经不再是必须付出的代价了。我们帮助众多软件企业和组织转危为安，现在也希望能够助你一臂之力。

在本书中，我们会介绍如何利用一套流程来创造商业价值，这套流程能保证

至少每 30 天就交付完整的软件功能模块；如何对你所需要的功能进行优先级排序，然后一一交付；如何根据期望功能跟踪已交付功能，以此来了解其商业价值，以及软件开发流程和组织整体上是否健康。用本书中介绍的工具及理念来武装自己，你将可以帮助你的软件企业快速掌握现代工程实践，开始交付你期盼已久的成果。

这就是 30 天软件开发。

致谢

感谢 Arlette Ballew 出色的编辑加工、Richard Narramore 的整体指导以及 Carey Armstrong 的倾情关注。如果没有他们的贡献，本书不可能付梓出版。

目　录

第一部分　为什么说每家公司都能在 30 天内开发出软件

你可能对你的软件公司感到沮丧，希望它能够更快、更灵活、更好地理解你的需要，并帮助你创造更多利润。在这一部分，我们首先找出令你沮丧的原因，然后想办法解决问题。

第 1 章　软件危机：错误的流程导致错误的结果 ····· 2

很多软件开发组织都在使用一种开发流程，而使用这种流程就意味着你肯定会遭遇浪费、无法控制的风险、不确定性、意外情况以及低价值。在这一章里，我们会研究为什么人们会选择这种流程，也会分析为什么这种流程注定要失败，最后再分享一些组织从失败中恢复的案例。

1.1　案例学习：FBI 的“哨兵”项目 ····· 3
1.2　错误的方法：预测性流程 ····· 5
1.3　错误的结果：项目失败 ····· 8
1.4　案例分析：PTC ····· 11
1.5　小结 ····· 14

第 2 章　Scrum：正确的流程产生正确的结果 ····· 15

有这样一个适合软件开发的流程。当开发人员使用它的时候，会立马提高生产效率、质量、价值、可控性、可预期性和满意度。我们会在这一章里看看这种流程是如何做到这一点的。

2.1　经验型流程实战 ····· 15

2.2 经验型流程真的能够解决问题吗 …… 19
2.3 人类实践源于经验主义 …… 24
2.4 尽管我们知道该如何做 …… 27
2.5 敏捷性 …… 28
2.6 小结 …… 28

第 3 章 你也来试一试：创建试点项目 …… 30

现在你已经对我们所宣称的更好的软件开发方法有所了解了。然而，过去也有很多人宣称他们的方法是最好的，并从你的口袋里赚到了很多钞票，却只为你带来了极小的改进，甚至没有任何改进。在这一章里，我们要向你证明我们所介绍的流程是可行的并且是免费的。

3.1 经验主义已经在组织中使用了 …… 31
3.2 一个试点范例 …… 32
3.3 这对团队成员来说可能是全新的工作方式 …… 42
3.4 小结 …… 44

第 4 章 我要做些什么 …… 45

到目前为止，你已经学到怎么才能做得更好，也有了切身体会。你为这样的结果感到兴奋，同时也知道如何向软件组织介绍新的流程。在这一章里，我们会介绍如何应用你的经验帮助你的试点项目取得成功。

4.1 实践可能性艺术 …… 45
4.2 创造透明的成长环境 …… 47
4.3 相信你的员工能做更多 …… 48
4.4 降低员工对确定性的期望 …… 49
4.5 小结 …… 50

第二部分 如何在 30 天内开发出软件

根据需求开发出更好的软件并不像过去那么困难。在这一部分里，我们将会介绍一套循序渐进的方法，帮助你从目前的状态过渡到让整个组织变得敏捷起来。

第 5 章　初试 Scrum……52

我们用于帮助你改进软件开发的秘密武器叫做“Scrum”。是的，就是英式橄榄球里的争球。在这一章里，我们将会讨论 Scrum 是如何工作的，以及它为什么有效。

5.1　组建 Scrum 团队并为 Sprint 做计划……53
5.2　开始 Sprint——向价值启航……53
5.3　进行 Sprint 评审……54
5.4　进行 Sprint 回顾……55
5.5　继续 Sprint……55
5.6　小结……56

第 6 章　在项目中应用 Scrum……57

软件开发中的大多数持续改进都是从项目开始的。你可以运用 Scrum 来进一步证明它的效果，或者在必须成功的、至关重要的项目中使用 Scrum。这一章里，我们会探索如何培训开发人员。

6.1　自下而上的隐形 Scrum……57
6.2　好处与收获……58
6.3　使用燃尽图管理工作进度……58
6.4　不要忽视复杂性：永远保持警惕……62
6.5　Sprint 的长度……63
6.6　下一章……68

第 7 章　创建 Scrum 工作室……69

成功通常能够带来更多的成功。随着越来越多的软件项目成功实施 Scrum，越来越多的人也希望加入到 Scrum 的行列中来。不考虑尝试变革整个组织，我们来看看如何从令人失望的现有部门中为 Scrum 开辟一片独立的天地。你将可以逐渐地从越来越多的项目和发布版本中享受 Scrum 带来的益处。

7.1　工作室是一个学习型的组织……69
7.2　工作室经理……70

7.3 培训和使用条款 …… 71
7.4 工作室的设施 …… 73
7.5 变革和难题 …… 74
7.6 用数字进行管理 …… 75
7.7 依赖于透明性的指标 …… 78
7.8 一个完成并且完整的功能增量 …… 79
7.9 一个类比 …… 83
7.10 消除技术债务获得可用的增量 …… 84
7.11 罪恶之源 …… 89
7.12 小结 …… 91

第 8 章 在企业中应用 Scrum …… 92

Scrum 在项目或者版本发布层面带来了初期的敏捷性，同时也带来了迅速响应机会和应对挑战的能力。为了获得最重要的收益，作为经验型流程的 Scrum，必须让整个组织都融入进来。这一章里，我们会探索如何去做，以及为什么有些方法无法长久，而有些却可以。

8.1 深入但短暂的改变 …… 92
8.2 深化并固化的改变 …… 94
8.3 Carbonite 公司的转型 …… 95
8.4 Carbonite 的改革之举 …… 95
8.5 结果 …… 96
8.6 Scrum 实施中无可争议的两个元素 …… 96

第 9 章 企业级转型：深化并固化改革 …… 98

你希望在任期内让你的组织变得更精益、更高效、更敏捷，甚至希望这些利益和潜在的成因能够在组织中持久沉淀并且变成组织的文化。我们会在这一章里探讨如何才能进行企业级变革，实现以上这些期望。

9.1 企业转型工程 …… 98
9.2 做好准备 …… 99
9.3 启动转型工程 …… 99

9.4　传播愿景和策略 …… 102
9.5　推向整个组织 …… 104
9.6　造成影响 …… 106
9.7　度量、评估并巩固成果 …… 106
9.8　巩固、推广并坚持 …… 107
9.9　小结 …… 108

第 10 章　用 Scrum 的方式实施 Scrum …… 109

我们设计 Scrum 就是为了解决像软件开发这样复杂的问题。我们发现 Scrum 是管理组织变革和复杂问题的强力武器，而且能够在透明性、减少浪费、风险控制以及可预见性方面获得好处。这一章里，我们将会探索如何在这方面运用 Scrum。

10.1　SeaChange International 用 Scrum 实施 Scrum …… 109
10.2　SeaChange 的破冰之举 …… 110
10.3　结果 …… 112
10.4　Iron Mountain 推广 Scrum …… 112
10.5　转型团队 …… 113
10.6　小结 …… 115

附录 A　术语 …… 116

我们慢慢地、循序渐进地介绍一些新术语，这份附录可以作为你的术语参考。

附录 B　Scrum 指南 …… 120

通过阅读这篇权威的指南，你将了解到 Scrum 中的角色、工件及事件。这是一篇 Scrum 的圣经。

附录 C　企业级敏捷攻略 …… 139

这份附录更详细地介绍了第 10 章所述的进行企业级变革的计划。

Part 1 第一部分

为什么说每家公司都能在30天内开发出软件

我们渴望帮助每一位希望创造出更具价值和可预测性的软件的组织领导者。软件行业正在经历巨变并日臻完善。过去软件开发具有很高的不确定性，风险大，浪费严重，这些司空见惯的情况现在已有所改观。在过去的20年里，我们帮助众多组织起死回生，获得了丰富的经验。在此，我们也想助你一臂之力，帮助你开发出风险可控、可预见、有价值、高质量的软件。

这么做的原因有两个。首先，你已经饱受软件行业的折磨长达40年了，并非心甘情愿，而是无可奈何。我们希望恢复那让人愉悦的良好合作关系。其次，软件不再默默无闻，它已无处不在，在现今社会发挥着越来越重要的作用。我们希望你能制造出可靠的软件。

希望能通过本书达成我们的目标。恳请你无论如何都不要放弃，因为你不再需要面对以前那样糟糕的软件结果，请持之以恒坚持下去。

在本书的第一部分，我们先研究一下软件开发为何曾如此糟糕。然后看看它是如何逐步改进的，以及推动改进的两个潜在动力是什么。接下来会展示如何试行我们的流程，怎样正确运作。第二部分将会介绍如何用更严谨的步骤使用我们的新流程，当然前提是你对试行结果满意。

第 1 章

软件危机：错误的流程导致错误的结果

无论是商业组织、政府机构，还是非营利组织，都可能需要通过开发、定制和使用软件来创造价值。假如没有软件的帮助，作为商业领导者，你很难甚至无法实现自己设定的业务目标。尽管软件如此有用，但是一直以来，软件开发都广受诟病，被认为不可靠、高成本、易出错。[①]这就使你陷入了一个两难的境地：你需要软件，但无法及时得到成本可接受、质量又可靠的可用软件。

实际上，斯坦迪什组织（Standish Group）在2011年的CHAOS报告中指出，2002年至2010年期间，超过半数的软件项目都被评定为存在缺陷或者是彻底失败，只有37%的项目是成功的（如图1-1所示）。斯坦迪什组织对成功项目的保守定义是：能够按照既定的预算，在承诺的期限之内交付所有需要的功能。而软件适应变化的能力、管理风险的能力以及软件的固有价值均未纳入评估范畴。

软件项目的成功率如此之低，你很可能会为那些牵涉到软件开发的重要项目忧心忡忡。软件行业耗时、成本高，又不可预期，早已让你大失所望。如果不是软件那么重要的话，你很可能早就停止在软件上的投资了。

① 2005年4月11日的弗雷斯特报告“企业软件开发无法满足速度和质量需求”指出，企业软件开发公司仍然让人失望：弗雷斯特在2004年秋季的一项调查表明，将近三分之一的受访对象对软件供应商交付定制软件的时间不满意，相同比例的受访对象对最终交付的软件质量感到失望，五分之一的受访对象对软件质量和交付时间都不满意。该调查的692位受访对象均为在IT领域具有影响力、掌控财政大权的人士。

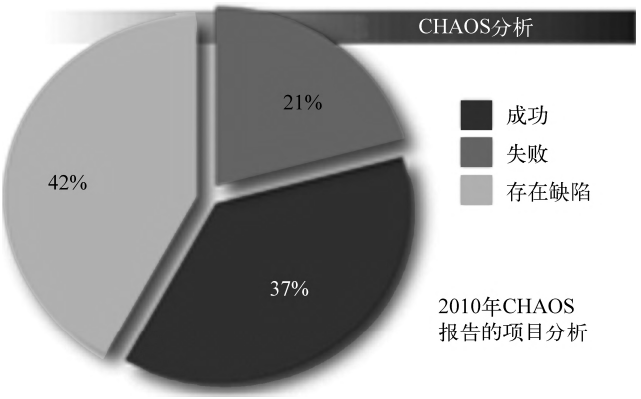


图1-1　传统软件开发存在风险

不过，面对这些问题的并不只有你一个，很多机构都有同样的困扰。例如，联邦调查局（FBI）的“哨兵”（Sentinel）项目最近也遇上了麻烦。结果，他们运用本书中描述的理念和流程，使项目起死回生。

这里关于“哨兵”项目的资料均来自美国司法部的检察长报告，并且对外公开。如果你认为这只是一个基于政府工作特性的个案，不具有普遍性，那么试想一下：如果一个庞大的政府机构都可以这样彻底地改进软件的开发方式，你的组织同样可以做到。

1.1　案例学习：FBI的“哨兵”项目

FBI针对每一项调查都建立一份档案，其中包含所有创建或者调用的调查记录。2003年，FBI决定将这些档案数字化，并对相关的流程进行自动化管理，这样探员们就可以迅速地比较各个档案，从中找到它们之间的联系。这个项目的名字叫做“哨兵”。

2006年3月，FBI启动了“哨兵”项目，目标用户是3万多人，包括FBI探员、分析师和行政人员。“哨兵”项目的最初预算是4.51亿美元，预期在2009年12月完工。根据FBI的最初计划，“哨兵”项目的开发工作会分为四个阶段。这个项目外

包给了洛克希德马丁（Lockheed Martin）公司，这家公司建议使用传统的软件开发流程。

到2010年8月，FBI已经花费了总预算中的4.05亿美元，但是项目只完成了四个阶段中的两个。虽然完成的前两个阶段的确帮助FBI改进了档案的管理系统，但是没有达到他们的预期效果。于是，由于成本和时间的超支，2010年7月，FBI决定停止洛克希德马丁公司在“哨兵”项目剩余两个阶段的开发工作。

到这个阶段为止，FBI一直在使用传统的开发流程，而现在，他们尝试引入新的流程来获得更好的成效。这个新的流程就是我们在20世纪90年代初创立的Scrum。那份指出只有37%的软件项目成功的CHAOS报告，同时论述了采用传统开发流程和采用敏捷（即Scrum）流程开发项目的结果会有什么区别（如图1-2所示）。从图中可以看出，采用传统开发流程（瀑布式）的项目中只有14%成功了，而采用敏捷流程的项目成功率则有42%。这些敏捷项目不仅满足斯坦迪什组织对成功项目的定义，而且能够更好地适应客户需求的变化，更有效地降低风险，并最终交付更高质量的软件。

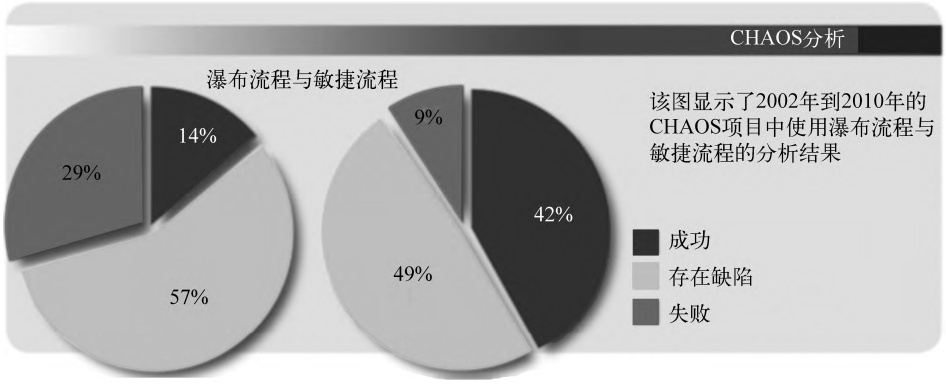


图1-2　敏捷项目的成功率是三倍

2009年，FBI聘请了新的首席信息官和首席技术官，他们都管理过使用敏捷流程进行软件开发的组织，于是决定看看敏捷流程能否带来帮助。2010年，首席技术官向司法部提出他希望改变“哨兵”项目的开发流程，声称新的流程将精简决

策过程，并且能够在预算内完成剩余的开发工作。FBI告诉当时的司法部检察长，他们相信可以在12个月内，利用剩余的预算完成“哨兵”项目。根据MITRE的审计显示，如果FBI继续沿用传统开发流程的话，他们将需要额外的3500万美元和6年时间来完成该项目。

于是，FBI将整个“哨兵”项目搬到了位于华盛顿特区的FBI大楼地下室，并且将项目的人数从400人减少到45人，其中15人是程序员。项目重新启动后由首席技术官亲自运营。他要求团队以30天为一个周期，每个周期交付一部分功能，每次交付的功能必须满足最终的功能性和非功能性需求。这是一个“一次发布”的软件。此外，每三个月FBI就将最近三次迭代的功能部署到一个现场试点上。

2011年11月，“哨兵”项目在重新启动新的流程后用不到一年的时间就全部完成了。软件首先部署到FBI的一些试点办公地，其余的办公地将在2012年6月前完成部署。事实是，FBI用了3000万美元和不到12个月的时间完成了“哨兵”项目，并且节省了90%以上的开支。

虽然FBI在“哨兵”项目的前期阶段投入了大量的人力物力，但是开发流程严重阻碍了项目的进展。而实施了本书所介绍的流程以后，虽然工作强度跟以前一样，回报却丰厚得多。如果FBI能这么做，为什么你不可以呢？

1.2　错误的方法：预测性流程

FBI在“哨兵”项目初期所使用的流程，就是我们所说的预测性或顺序式的设计流程。实际上，在2005年之前，大多数软件项目仍然使用预测性流程。并不是说这种预测性流程就一定不好，它在特定场景下会更适用，并能取得成功。然而，这些场景只是特例而非常态。例如，已经建立了完整的愿景，愿景中的所有需求已经定义，并且已经策划出实现愿景的详细计划，这时就可以使用预测性流程。但是，只要与原始愿景、需求或者计划略有偏差，就会为项目带来巨大的风险。然而由于商业需求和技术的日新月异，可以说这些因素的变化都

是不可避免的。于是，就像斯坦迪什组织在其报告中所述的，86%使用预测性流程的软件项目都未能成功。实际上可以认为，预测性流程的使用是导致软件项目失败的最常见原因。

与我们合作过的组织都在想尽办法提高软件项目的成功率。他们担心软件项目失控，来向我们求助。现有的流程让他们尝到了失败的滋味，却无计可施。这些软件开发流程浪费了他们大量的人力物力。但由于软件是其主要竞争力来源，他们不得不继续这些项目。

下面是高管和经理们经常遇到的问题。

(1) 版本发布所需时间越来越长。“提交给客户的每一个版本，其发布所需要的时间、人力和成本越来越多。几年前，一次发布可能需要18个月。现在，同样的一个版本发布需要24个月来开发、打包和部署。尽管如此，我们仍然感觉时间仓促，压力很大。我们觉得投入越来越大，而产出却越来越少。”

(2) 无法按时发布。“在我们向客户和市场承诺之后，客户或者潜在客户都按照我们承诺的发布时间制订商业计划。他们需要我们在承诺的时间点，及时交付承诺的功能。然而，我们通常在最后时刻让他们大失所望。他们的计划被打乱，损失了金钱，失去了客户的信任。结果是，我们可能再也无法从他们那里拿到新的项目，他们显然也不会为我们介绍新的客户，他们自己也会寻找新的开发商。”

(3) 在版本发布的最后阶段，软件达到稳定状态需要的时间越来越长。“我们和软件开发组织有明确的约定，确定了固定的交付日期。虽然他们在设定的日期之前达到了他们所谓的‘代码完成’或‘代码冻结’状态，但是软件根本不能用，无法实现所需的功能和性能，让人完全无法接受。我们甚至无法将‘测试版本’交付给小样本客户试用以获得反馈，因为软件中的缺陷实在是太明显了，试用客户拒绝参与测试。我们需要额外的九个月时间来完成版本发布。即便是那样，我们还是没有十足的把握，仍然需要很多帮助，并为各种事情向客户道歉。”

(4) 制订计划的时间太长，而且计划得不准确。“我们发现，由于我们没有在

项目开始之前把计划做得足够好，导致发布版本的时间过长，错过了发布日期。我们没有完全确定并分析清楚需求，而且我们的预估包含了过多猜测。为了解决这些问题，我们不得不花费更多的时间来做计划。在这期间，我们不断产生新的想法，评审计划的过程中也会发现需要重做或澄清的地方。结果是，现在我们花在计划上的时间更多了，然而进度落后和软件难以达到稳定状态的问题仍然非常严重。尽管我们为做计划大费周章，但在开发期间仍会遇到计划期间未能或无法预见的变化。”

(5) 在版本发布中期很难做更改。“目前的开发流程无法适应变化。我们在开始开发时投入了大量的时间做计划，并且断定所有需要的工作都列入了计划中，但后来经常会发现有些非常重要的部分或新的特性必须添加到当前版本中。为了进行这些更改，我们不得不调整已经完成的部分。而这种调整是非常困难的，因为在软件中引起的连锁反应是难以想象的。尽管调整是必要的，但在版本中调整比在一开始调整要多花一百倍的时间。但是我们又能怎么做呢？如果不在当前的版本或者项目中做出调整，恐怕就要等到两年后的新版本才有机会了。”

(6) 质量持续恶化。“我们知道不该向开发人员施压要求按期交付原定和更改后的功能。但是当我们的业务遭受计划、延误、变化问题的三重打击的时候，我们只能告诉开发人员，必须咬紧牙关在限期内交付所有计划中的东西。每当开发人员听到这种指令时，他们通常会用降低软件质量或者减少软件适用性测试的方式来应付我们。结果可想而知，要么回到版本稳定化阶段，要么交付质量差的软件，使组织的声誉受损。”

(7)“死亡行军”损伤士气。“我们现在管理员工的方式很严苛，但这也是不得已而为之。我们必须完成组织的目标和业务，因此项目中的每个人都需要加班，而且周末也要上班。这会使他们的家庭关系和健康受损。后果是，我们很难招到顶级的开发人员，而且眼睁睁看着公司最优秀的开发人员跳槽到其他组织。此外，现有的员工士气低落，工作效率随着工作时间的延长而降低。”

这些例子已经足以让高管和经理们垂头丧气。尽管整个软件行业在20年间投

入了大量的努力和金钱，但截至20世纪90年代初，仍然未在软件项目的成功开发上有所突破。我们在本书中介绍的流程，正是解决这些问题的良方妙药。

1.3 错误的结果：项目失败

传统预测性软件开发流程的使用是导致如此之多项目失败的罪魁祸首。预测性流程也叫瀑布式流程，其可行性依赖于项目计划的准确性和执行的严格性。它必须满足以下条件。

(1) 需求不会变更，而且被完全理解。需求的任何变更都会使计划改变，从而产生连锁反应，最后的结果通常是导致已经完成的工作作废。不幸的是，在典型的软件项目中，35%以上的需求都会变更。一开始，客户会尽力确定所有需求。但商业环境瞬息万变，加上客户对自己实际需求的理解不完整，而且很难在成品完成前完整地描述预期系统，因此需求的变更就不可避免了。

(2) 技术能够正常使用。在软件中使用的所有技术都能按照最初计划的那样可靠地工作。不幸的是，项目中经常会引入做计划的人从未使用过的技术。可能是单独使用，也可能和其他技术组合使用，或者是不同技术用于同一用途。另外，技术标准有时也会在项目期间改变。

(3) 员工像机器一样可预测并能可靠地工作。计划中包含一系列相互关联的任务，每一项任务都需要专业技术人员根据特定的输入在给定的时间完成。然而一旦需求发生变化，就会产生连锁反应，所有任务都有可能受到波及。更重要的问题是，人并不是机器！人有情绪起伏，有技术水平差异，还有态度和智力的差别，不可能完全按照定义执行任务。因此到了最后，任务的执行情况可能与计划差异较大。

软件行业已经意识到上述这些困难，而且多年来一直试图增加计划方面的投入以解决这些问题。有时做计划的时间甚至和实际开发的时间一样长，在搜集需

求、定义架构以及细化工作计划方面做了大量的工作。

只有计划基于精准且保持不变的信息，前期的所有投入才有价值。因此，这种方法只有在需求完全清楚而且相对稳定，计划也不会变化的情况下才是有效的。假如实际情况并非这样，预测性流程就会失败，因为预测性流程并不适合解决未知和意外的问题，而是用于优化约束问题的。

很多传统制造企业成功地实施了预测性流程模型。这些前期的研究工作对后续重复性地执行计划是有好处的，如生产出一辆辆汽车或者一个个烤箱。然而，对于软件开发来说就不一样了，因为软件开发的计划只执行一次。预测性流程适合制造业的原因是一个流程会制造出大量的产品。而这恰恰是它不适用于软件行业的原因，因为软件开发的一个流程只会生产出一个产品。

斯黛西图（Stacey Graph）是用来评估工作中的确定性和可预测性的有效工具。[①]斯黛西图用于度量不同维度的工作的确定性和不可预见性，并标示出工作范围。我们用它来为软件开发中的三个维度建模，这三个维度分别是需求、技术和人，如图1-3所示。

我们可以把软件开发项目划分成下面三个方面。

- 需求。无变更风险时确定性最高。随着不明确因素、涌现式描述和可预见性变更的增多，确定性降低。
- 技术。所用技术为人熟知时确定性最高。随着开发及运营技术复杂度的提升，不同的技术在不同的软件开发和发布阶段通过接口交互，确定性随之降低。
- 人。确定性随着人员数量的增多而降低。当人员数量超过4个或者5个，甚至达到上百个并经常改变时，确定性不断降低，因为不同的人有不同的意见、态度和情绪。以团队或分组方式工作时，成员间的互动和工作的不可预见性是巨大的。

① 引用自R. Stacey的*Complexity and Emergence in Organizations*。

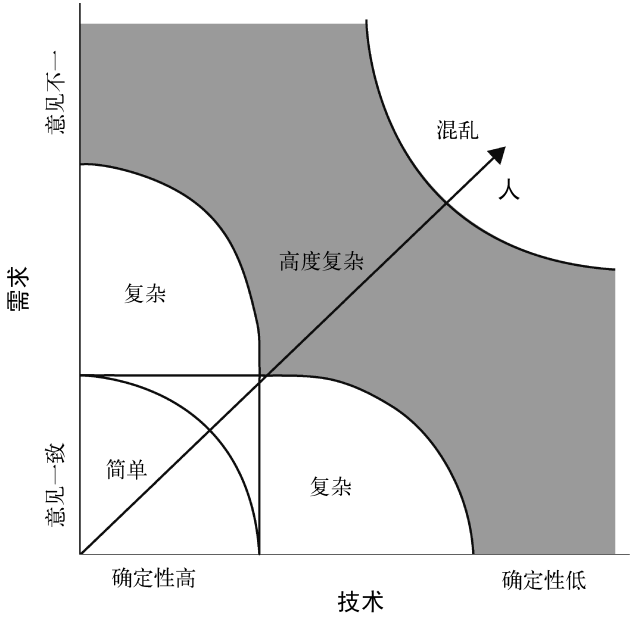


图1-3　斯黛西图

使用斯黛西图，可以看到软件开发项目是复杂的，有时甚至是混乱的。瀑布式开发和传统软件开发所基于的预测性流程只适用于简单的重复性工作。现在你可以根据收益率（成功率）明确地判断你的流程是否符合需要。如果预测性流程适合你的项目，那么收益率（或者说项目的成功完成率）将高达99.99%。然而，根据斯坦迪什报告，使用预测性流程的软件项目的成功率只有14%。大多数公司无法在如此低的成功率下生存。试想一下，如果通用汽车生产的每七辆车中就有一辆报废，就是14%的成功率。

预测性流程并不适合解决软件开发中的问题，因为软件开发是复杂的，而不是简单的过程。我们可以断定，将预测性流程用于软件开发项目是令我们失败的根本原因。依据是，自从开始应用Scrum以后，软件开发项目的成功率提升了。

人们有时候将工程或桥梁建造等同于软件开发。桥梁建造等工程规范在斯黛西图中位于简单和复杂之间的区域。标准化流程只能使这种工作变得更复杂。工程建造中一共有三种类型的标准化。首先，牛顿定律解释了物体间是如何相互作

用的。其次，采用的标准化材料，如木梁、金属支柱和扣件，都有标准的大小以及已知的特性。最后，所有类型的结构都有官方规范和检查标准。而在软件开发中却不存在如上这些标准。而且，随着软件行业的日新月异，这种没有标准的情况将不会改变。

1.4　案例分析：PTC

PTC（Parametric Technology Corporation）是一家拥有5000名员工的跨国公司，从事产品生命周期管理软件的开发。其产品基于CAD/CAM（计算机辅助设计/计算机辅助制造）系统，帮助像雷神、BAE系统、空中客车等世界顶级的工程组织管理大型系统的开发，例如空中客车A380的开发。其主要是通过监控所有部件、配件和子配件的配置来进行管理。

2005年，PTC遇上了使用预测性流程有可能遇上的所有麻烦。

(1) 版本发布所需时间越来越长。发布一个版本的时间从18个月增加到24个月，而且看起来当前版本可能需要更长的时间。

(2) 无法按时发布。交付延误长达9个月，而且情况逐步恶化。那些依赖于重要功能准时交付的客户为此感到很不高兴。

(3) 在版本发布的最后阶段，软件达到稳定状态需要的时间越来越长。软件稳定花费的时间至少占了延误期的三分之二。

(4) 制订计划的时间太长，而且计划得不准确。每个版本的计划期都长达6个月，但计划仍然不够准确，经常需要改变。

(5) 在版本发布中期很难做更改。尽管无法确定项目延期以及软件的稳定和质量问题是由项目开始之后的变更引起的，但是大量的变更确实发生了。

(6) 质量持续恶化。这是相当严重的问题，而且情况越来越糟。

(7)“死亡行军”损伤士气。PTC很难招聘到高素质的人员。

PTC的开发部门使用的是瀑布式流程。为了使其更好地工作，他们尝试把需求弄得非常清楚。需求都被记录在一个详细的功能规格说明书里，只有在最终确定后才会让开发部门知道。在确定需求的这段时间里，开发人员并没有什么事情可做。他们要么修复程序错误，要么就因为极度无聊离职了。而测试团队在整个产品完成之前无法进行测试，因此能够进行测试的时间很短。由于发布日期的压力，测试团队被迫发布没有经过充分测试的产品。

Jane Wachutka是PTC Windchill产品开发部门的新任副总裁。刚入职时，她尝试了PTC的瀑布式流程，结果碰到了所有常见的问题。在以前的工作中，她曾引入很多类似帮助FBI完成“哨兵”项目的非瀑布式流程。在这种流程中，一个项目包含一个或多个迭代工作，每一个迭代不超过30天。开发人员被分成多个小组，在每个迭代中选择高价值的需求进行开发，然后将它们作为增量集成到可用的软件中。所有团队开发的增量集成为一个完整的可用增量。在接下来的每个迭代中，都将开发新的增量，并添加到之前的增量中。

Brian Shepherd是PTC产品开发部门的执行副总裁。当Jane在2007年引入新的软件开发流程的时候，Brian曾经有过顾虑。当时，Jane承诺如果Brian允许她引入新的流程，她保证能让开发人员更早地开始工作，降低开发人员的流失率，让测试人员更早地参与工作，在产品未经过完整测试且质量过关之前绝不发布。Jane还强调，可以让开发人员在需求文档还没完善的时候就开始开发，这样产品管理团队可以在开发期间经常检查和试用开发完成的部分，并给予开发团队反馈。于是，Brian同意引入了一套新流程——Scrum敏捷流程，但是同时他也警告Jane：“别搞砸了。”

当Jane第一次告诉她的团队要使用新的流程进行开发的时候，他们也有所顾虑，尤其是开发团队的成员。他们仍然希望开发过程的每一步都是精准的，能够确保满足需求。然而，随着对新流程的日渐熟悉，产品经理们不再试图将完美的需求文档交给开发团队，而是允许开发期间涌现新的需求。由于PTC现在能够在

30天内开发出完整的功能，因此开发人员能够在任意适合的迭代中和客户直接交流。他们能够更直接地了解需求，以及如何更好地实现它们。客户也发现了这样的改变，开始在每个迭代中和开发团队一起工作。他们积极参与定义功能，得到了真正想要的功能。

产品管理团队拥有持续变化的三个需求集，分别是为三年、两年和一年的需求集。三年的需求集包含了总体愿景，描述了重要功能。两年的需求集中定义了这些愿景将在哪些版本中具体实现。本年度的需求集中包含了前6个月中每个30天迭代的需求，以及后6个月的目标路线图。每年的需求集都比前一年拥有更多细节。开发人员依照当年的需求集进行工作。他们和PTC的客户协作定义需求细节。整个组织变成了一个富有创造力和生产力的智囊团。

两年之内，Jane对Brian的所有承诺都兑现了。版本发布时间从之前的24个多月降到了12个月，而且交付的产品质量很高。到2011年，PTC已经发生了翻天覆地的变化，无论是对内部还是客户都是透明的。意外情况很少发生，客户知道在什么时候能够得到什么。到了2012年，缺陷率已经几乎降低为零。新的特性、用户界面和工作流功能都被加入到了产品中。整个产品也经过了彻底检查，安全性足以抵御外来的风险。最后，预算和人员需求都降低了超过十个百分点。Brian为软件产品开发部门建造了新的办公室。新的办公场所满足了新流程所需的透明度。现在每个人都在一个开放的空间里，没有独立的办公室，而且所有的墙都是玻璃的。

最近，PTC的首席执行官Jim Heppelmann听取了下属经理们关于提高年度预算的讨论。最后，他打断了他们，并让每个人感谢Jane的部门在提升质量和增加功能的同时降低了软件成本。正因为这样，才为其他部门腾出了提高预算的空间。

有一次，一个以色列客户与Jane和Jim进行电话会议，评估PTC公司的产品。Jane告诉对方的首席执行官，雷神公司正在其全球的子公司中使用PTC的产品，并建议他向雷神的高管咨询，因为她清楚地知道，雷神公司不仅对PTC的产品非常满意，还惊叹于PTC新的流程能够避免意外情况的发生。他们能够与PTC

实时合作，随时调整自身的计划，甚至还采纳了PTC的软件开发流程。Jim插话说，Jane忘记提及上一次发布的版本，那是PTC交付的最漂亮的一个版本，并且主要归功于Jane改变了开发流程。

1.5 小结

软件开发在过去非常容易失败，其根本原因就是使用了预测性流程来做复杂的工作。当Scrum这样的经验型流程被采用后，软件项目的成功率会大幅提高。

因此，在30天内开发出可用的软件功能是完全有可能的。不要听你的开发人员说这不可能，因为从21世纪伊始，已经有成千上万名开发人员成功做到了。也许软件产品仍然很庞大，但它可以被切分为几个小模块，以30天为周期逐个开发。

第2章

Scrum：正确的流程产生正确的结果

在上一章中，我们发现正确的软件开发流程是经验型流程。现在，让我们来看看经验型流程是如何工作的，应该如何运用它进行软件开发。本章我们将运用多年来研发的敏捷软件开发流程Scrum来探索经验型流程。

2.1 经验型流程实战

在经验型流程中，信息是通过观察而不是预测获取的。经验型流程是解决复杂问题的良方妙药，在这种情况下，未知的事情比已知的事情多。经验型流程可行的前提条件有下面两个。

(1) 检视和调整。我们必须经常检视当前的进度，这样才能够调整下一步的计划，从而得到最好的结果。检视和调整的频率取决于我们愿意承担多少风险。未知的事情越多，我们就越容易偏离目标。偏离目标越远，就会浪费越多的资源来重新定位目标，放弃无用的工作，重新开始。

(2) 透明性。我们必须从最终目标的角度进行检视。比如我们需要开发具有某些特性和功能的系统，那么我们要检视的对象一定是某种特性、功能，或者是两者各自的子集。

假如使用预测型流程，规划的需求可能需要花费数年才能实现。但是我们知道，在软件项目开发中，开发时间如此之长将增加过多的风险，并造成巨大的资

源浪费。因此，我们将开发周期缩短至30天或更短（我们将在后面讨论缩短周期的意义）。在第一个30天后， 我们可以检查结果，并据此决定下一步的工作，适时地调整路线，实现既定的目标。

在开始开发之前，需要先有一个如何利用软件创造价值的想法或者愿景，知道如何有效地运营，如何开发出有价值的软件，还可以清楚地描述出软件需要实现的功能及必须满足的需求。相对于这些而言，其他事情可以暂时放到一旁，以后再考虑。我们需要知道的，包括至关重要且明确理解的内容，也包括一般相关且不甚明确的内容。

接下来列出包含所有想法的清单，这个清单叫做需求待办列表（如图2-1所示）。我们按照优先级对需求进行排序，把最重要的需求排到最顶部。这个列表记录了我们的想法，可以随时更新，根据需要增加、修改或删除其中的项目。

银行业的部分产品待办列表						
业务范围	操作	平台	活动	产品待办列表项	优先级	大小
信托	……					
企业银行	……					
个人银行	柜台职员	抵押				
		储蓄	存款	客户可以跨账号存款	33	13
				客户可以使用新式自动柜员机存款	42	21
			取款			
		支票				
	平台	IRA	档案状态			
		401K	个人信息			
		抵押	位置			
		个人贷款				
		储蓄				
		核算				

图2-1　根据业务操作划分的需求待办列表

首先，需要确定我们的想法是否可行，能否在30天之内开发出有用的软件，

并验证后续的工作。

我们召集一个软件开发小团队，与其分享我们的愿景和初始需求。我们和这个团队协作确定最重要的需求。虽然整个系统可能是非常庞大的，但是我们只关注我们正好能够可能和需要完成的部分。同时，我们也希望能够尽快将部分愿景变成可用的软件。

我们会问开发团队，在接下来的30天之内他们可以交付多少个可使用的完整功能。我们从最重要的需求开始讨论，而开发团队可能会带入其他的问题，如软件的稳定性问题。我们会详细讨论这些需求，帮助开发团队找到最好的开发方法。虽然我们不是开发团队成员那样的专业开发人员，但能够帮他们澄清需求，做出选择。

让我们来定义一下本书中提到过的一些术语。

- 迭代。所谓迭代，就是重复一系列步骤或某个流程，一般是为了实现一个预期目标或者结果。流程的每一次重复就叫做一次迭代，一次迭代所产生的结果被用作下一次迭代的开始。对你来说，第一个30天就是第一次迭代。
- 频率。频率指的是迭代的长度。高频率的迭代通过持续检查进度来有效控制浪费、降低风险。频率的范围最好在一个星期以上，一个月以内。
- 增量。增量就是不断增长的整体的一部分。开发流程中一次迭代所产生的功能结果就是增量。增量随着迭代的增加不断增长，直到变成我们所需的有价值的系统。
- 透明性。增量必须是完全完成并可用的，不能有任何工作落下。未完成的工作或原形都不具有透明性，因为我们无法判定其完成度是多少，剩余的工作还有多少。
- 迭代增量式开发。这是通过一系列迭代进行软件开发的方式。每个迭代都会基于之前的所有增量产生出新的功能增量。迭代会持续进行，直到完成目标，实现最大价值。

于是，我们开始了第一次迭代。开发团队将我们的需求转化为功能增量。每次迭代都从做计划开始，之后开发团队按照计划进行开发，最后所有人共同检查这次迭代产生的软件增量。

可能需要若干或者很多次迭代才能开发出满足我们需求和愿景的系统。每次迭代的时间都是固定的，也就是说，我们为每次迭代分配和使用的时间长度是相等的。每次迭代都会产生潜在可用的软件增量（如图2-2所示）。每个软件增量必须是没有任何遗漏的完整功能。每次迭代产生的软件增量都会作为下一次迭代的起点。

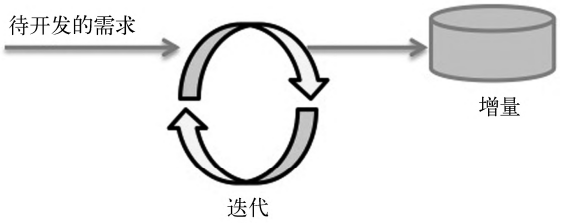


图2-2　一次迭代产生一个透明的增量

在一个迭代完成之后，我们就可以根据实际情况指导开发团队，甚至改变最初确定的方向。实际上，这种情况发生的概率很高。最初，我们拥有一个愿景或者好的机会。然后我们让开发团队创造软件来实现其中的重要部分。一次迭代后，我们得到了第一个增量。于是我们开始思考如何利用它，向其中添加什么使其更加实用。在某些学科中，这被称做“中期修正”。而在经验型流程里，这种修正在每次迭代都会发生。

在愿景的激励下，开发完每个增量后，我们都会思考更具创造性或更独特的方式实现愿景。有时候，这会引发我们和开发团队的对话，讨论如何在下一次迭代中获得最大价值以及下一次迭代应该做什么。我们乐于拥抱变化。

也许我们会发现我们的愿景并不现实。在实现愿景的最佳时机来临时，技术手段还没准备充分。也许我们对结果不满意，或者觉得成本太高了。这些情况有

可能迫使我们在这时停下前进的脚步。在找到更可行的愿景之前，我们不再花费一分钱。杜绝浪费也是成功项目的要素之一。

有时侯只需要一次迭代就能够开发出可用的软件，在这个基础上我们让开发团队开发更多的功能。我们充分利用机会，通过一次次迭代开发出越来越多的功能。每次迭代开发出来的增量都构建在之前所有增量的基础上。一旦开发出的产品满足了我们的需求，就可以发布供人们使用了。图2-3展现了包含多次迭代的版本发布。

我们将经验型流程作为软件开发流程。在每次迭代结束的时候决定下一次迭代要做的事情，并且持续将实现愿景作为目标。我们会评审已经完成的任务，并根据推测的成本和交付日期决定是否继续。这种流程叫做“迭代增量式流程”，也就是Scrum的基础。到目前为止，我们已经描述了这种流程是如何工作的，以及它为什么被称为30天软件开发。现在，来看看它是如何解决瀑布式流程（或者说预测型流程）中的问题的，它能否解决之前提到的问题，甚至更多的问题。

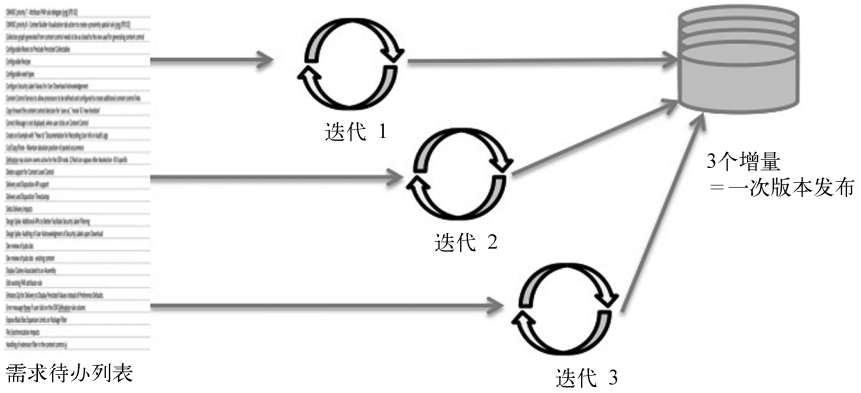


图2-3　多次迭代产生一个叠加功能增量

2.2　经验型流程真的能够解决问题吗

经验型流程真的能够解决瀑布式流程中的问题吗？让我们来逐项分析瀑布式

流程中的痛点吧。

- 瀑布式流程问题之一：版本发布所需的时间越来越长。在经验型流程中，版本是由一系列增量集成在一起组成的，这些增量通过一次次迭代按顺序开发。我们还可以在任意时间停止迭代。一旦发现产品的价值已经达到最大，尤其是发现软件里过半的功能很少被使用的时候，就可以停止迭代。我们还可以在到达交付期限或者预算上限的时候，停止迭代并发布软件。我们只会开发并积累有价值的增量。
- 瀑布式流程问题之二：无法按时发布。在经验型流程中，版本的发布最多延迟30天，因为每次迭代的最长期限只有30天。当到达交付期限的时候，我们就可以交付累积的增量。由于我们不会在迭代中开发低价值的功能，因此可以比以往更早地发布完整的系统。在传统开发流程中，常用功能的数量还不到功能总数的一半。而在经验型流程中，我们根本不会在不常用的功能上花时间。
- 瀑布式流程问题之三：在版本发布的最后阶段，软件达到稳定状态需要的时间越来越长。在经验型流程中，每次迭代所产生的增量都是完整和可用的，并且接下来的增量都会包含之前所有的增量，因此所有增量都是完成且可用的。也就是说，没有软件发布之前的稳定化阶段，因为软件一直保持稳定。
- 瀑布式流程问题之四：制订计划的时间太长，而且计划得不准确。在经验型流程中，初始计划简化成为设定目标，并确定达到目标所需的高价值功能和特性。然后预测交付日期和成本。第一次迭代之前的计划时间通常只有瀑布式或预测型流程的20%。我们只为即将开始的迭代做详细的计划，这被称为“适时计划”。另外值得注意的是，需求是涌现的。我们在检视每个增量结果的时候，都会为下一次迭代调整最佳需求。
- 瀑布式流程问题之五：在版本发布中期很难做更改。版本发布中期的概念在迭代增量式项目中已经不复存在了。在每次迭代开始前，都能够以最小的代价发现和提出需求。

- 瀑布式流程问题之六：质量持续恶化。在经验型流程中，每次迭代产生的增量都是完整且可用的，因此必然已经通过质量要求。而后续的增量同样要达到相同的质量要求。也就是说我们再也不必在项目最后的稳定化阶段为了赶上交付期限而牺牲质量，因为在开发过程中质量已经得到了保证。
- 瀑布式流程问题之七：死亡行军使员工士气受挫。版本最后的稳定化阶段已经不再需要，因此加班的“死亡行军”也随之而去。

正如我们所看到的那样，基于经验型流程的迭代增量式开发有效地解决了传统软件开发中的种种问题。为了满足所有企业的需要，我们需要懂得如何管理这些项目。接下来我们将探讨这个问题，并在第6章详细分析。

要管理这样的项目，只需要用到三个变量。首先是（A）需求，也就是预想中软件所需要的功能。其次是（B）时间，我们使用30天作为时间单位。最后是（C）完成的工作，用已交付的可用功能来度量，也可以看成是在任意30天内累积完成的（A）。

你可以创建如下图表来进行项目管理。

(1) 以需求待办列表作为y轴即垂直轴。每个需求所需要的工作量都用单位工作量来衡量。假设我们有5个需求，它们分别需要2、3、5、3、8个单位工作量。这些需求在y轴上达到了21个单位工作量。你可以根据需要对这些需求进行排序。我们暂且假设自上而下的顺序仍然是2、3、5、3、8。

(2) 以时间作为x轴即水平轴。以一次迭代的长度，即30天作为单位长度。

(3) 根据过往的经验，我们预期开发团队能够在每次迭代完成5个单位工作量。一旦团队正式开始工作，就能够知道其实际工作效率，当然这种预测是基于过去的经验的。我们预期团队可以在前4次迭代（5，5，5，5）完成20个单位工作量，然后在第5次迭代完成剩下的工作量。

(4) 在每次迭代结束时计算已经完成的工作量和可用功能总量。我们计划在

第一次迭代里完成前两个需求，分别是2个和3个单位工作量。然后在第二次迭代里完成工作量为5的下一个需求。在这之后，我们通常会改变下一步的计划。通过检视前两个增量，我们经常会发现一些未曾预料的新需求，需要在下一次迭代里实现。如果没有新需求的话，就继续按原定的计划进行。然而，就算需求或者计划在迭代结束的时候改变，也不会带来什么损失。增量的度量单位和y轴上的需求一致。

(5) 开发团队在前三个迭代中分别完成了3、5、5个单位需求。图2-4展示了这三个迭代的成果。

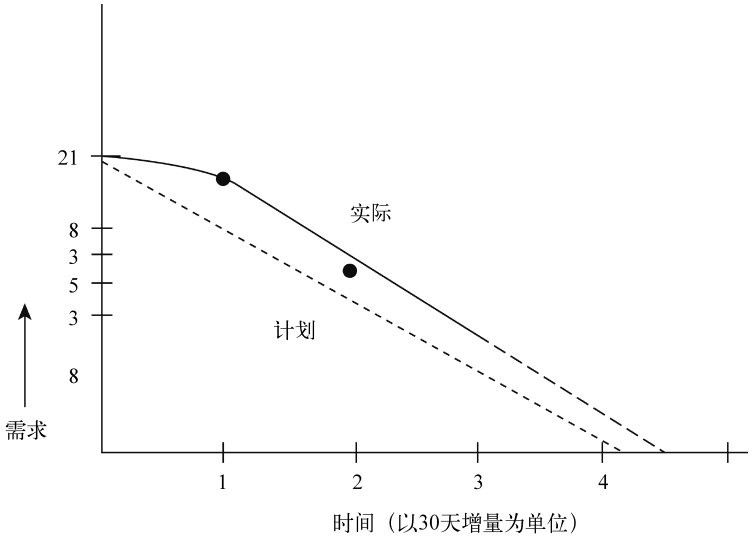


图2-4　发布燃尽图

根据最初的计划或者预测，项目需要21个单位工作量，预计每次迭代完成5个单位工作量。图2-4中的虚线表示预测的工作计划。可以看出，在计划中，第五次迭代开始不久之后，就能够完成所有功能。

然而，实际的情况是，第一次迭代完成了3个单位工作量，第二和第三次迭代分别完成了5个单位工作量。在图2-4中用实线表示实际的工作进度。如果我们按照当前的实际进度预测接下来的工作进展（如图2-4中实线的延长线），如图所示，

我们将在第五次迭代的中期而不是开始完成所有工作。当然，这仍然是我们的预测，无法确定。经验理论就意味着在工作实际完成之前，无法确定我们究竟可以完成多少工作。我们预期在第一次迭代能够完成5个单位工作量，却只完成了3个，因为所用的技术没有预期的可靠，有一个需求不够清楚，还有一位开发人员休了几天病假。我们在第一个迭代结束时检视了实际进度，发现投资回报率依然可观，而且在第一次迭代中发生的问题应该不会再出现了。基于这样的预期，我们冒险进行了下一次迭代。在每次迭代结束时，我们都会进行检视和调整。

经验主义对于以下方面大有裨益。

(1) 管理。在每次迭代结束时，能够准确知道有多少需求和哪些需求已经完成，以及哪些功能已经可用。可根据以往的进度制订将来的计划，并估算出可能完工的日期。在制订计划的过程中，可以清楚地意识到计划有可能在下一次迭代结束时改变。

(2) 控制。如果信息显示当前的进度比预期要慢，可以减少剩余功能的数量或者范围。例如，在第二次迭代结束时，还有13个单位的需求要完成，可以将剩余需求减少到10个单位。如果在接下来的两次迭代中，开发团队都能够保持5个单位需求的开发速率，那么将在第四次迭代结束时完成所有的功能。

(3) 可预期性。有时候预测有可能是错误的。实际完成工作的日期有可能比预期晚几个星期。你有可能在第一次迭代结束时有延期的担心，在第二次迭代结束时觉得可能性更大了，在第三次迭代结束时基本可以肯定了。这时候，任何需要使用这些功能的人就可以开始同步调整他们的计划。类似地，项目的预算也应该尽早地修改并确认。

(4) 风险管理。开发团队在前3次迭代中分别只完成了2个单位的功能需求。在第三次迭代结束时，可以预测所有工作将会在第十次迭代的中期完成。如果原来的预算是10万美元的话，根据新的预测，预算将超支15万美元。如果25万美元的投资回报率过低的话，就可以在第三次迭代结束时取消该项目。

2.3 人类实践源于经验主义

由于经验型流程使得什么工作可行、什么工作不可行变得透明，我们能够快速地学习这种开发方式，并为其制订一系列最佳实践。这些实践部分来自学术准则，部分来自团队自身的体会和实践。

总的来说，我们发现小团队最适合进行这种迭代增量式的软件开发工作。这里所说的小团队，一般来说应该由3~9人组成。团队总体应该拥有将需求转化为功能增量所必备的技能，从而实现你的愿景。根据开发的软件类型，团队中的成员应该具备编程、测试、设计、分析、编写文档、设计架构等技能。我们希望通过调整团队结构、提高工程实践能力和建立规范来塑造团队特质，提高团队的生产率、质量、创造力和持续改进的能力。

我们对高效软件开发团队的见解，大量地参考了竹内弘高和野中郁次郎的研究结果[①]。他们曾经在哈佛大学进行关于团队流程的研究。他们调查了拥有较高目标的自主团队，其团队成员积极地相互学习，以短期迭代为主要工作方式。这些团队之间紧密的协作能够缩短知识更新周期，从而有助于创新和更快地响应市场，获得高质量的结果。这些团队让他们想起了英式橄榄球，于是他们把这种项目管理方式叫做Scrum——英式橄榄球中球出界后重新开始比赛的仪式。

基于竹内弘高和野中郁次郎的研究，我们发展了以下实践来补充经验型软件开发流程的结构。这些实践能够打造出拥有出色创造力、质量、工作效率和士气的高效团队。

- 尊重每位员工。在一些公司里，员工被当成孩子。他们的创造力会被削弱，每天只能被动地接受别人指派的工作。如果要让员工对自己的工作拥有自

① 引自1986年1~2月，竹内弘高和野中郁次郎发表于《哈佛商业评论》的“新新产品开发游戏”（The New New Product Development Game）。

豪感以及参与感，必须营造一个充满鼓励和尊重的工作环境。Scrum的设计能确保员工受到尊重和赞赏。当然，我们不是第一个想到在Scrum里运用这种实践和想法的人。行业中已经存在很多最佳实践了。而Jeff非常关注Scrum软件开发环境中“人”的因素。

- 营造宽松氛围。开发流程开始时，管理层制订富有挑战性的宽泛目标或者战略方向。管理层不应该制订详细的产品和工作规划，而应该留给开发团队充分的自由和发挥空间。制订富有挑战性的需求能够有效地在团队内部创造动态张力。
- 自组织项目团队。由团队自己决定如何实现管理层的目标。这么做的目的是为了防止团队依赖于外部的意见，从而培养团队自我组织和管理的能力。当团队展现出自我管理、自我超越、相互学习这三方面能力的时候，就证明他们已经成为了真正的自组织团队了。团队中需要自我管理是因为管理层只提供指引、资金和支持，而不会对团队进行过多的干预。从某种意义上来说，管理层扮演的更像是风险投资人的角色，只提供金钱的支援而不会对团队指手画脚。在这种情况下，团队会不懈努力做到更好，并且为达到极致而不断探索。
- 相互学习的团队。不同职能的团队在同一地点工作，能够培养更高的效率、质量和创造力。团队成员互相协作，专业技能的界限渐渐地变得模糊。实际上，有些公司规定每个团队成员必须拥有两项专长（例如能运用两种或以上编程语言进行编程和测试）以及两个不同领域的知识（例如设计和市场营销）。团队成员间的紧密互动使团队开始形成自身的脉搏和节奏，产生韵律性的创新和效能。
- 开发阶段之间的重叠[①]。由于取消了“线性”顺序的工作方式，团队可以适应开发过程中各种障碍带来的“震动”或者“噪音”干扰。当遇到瓶颈的时候，团队应该能够绕开它，而不是突然就中断工作了。重叠开发阶段的理念

① 在瀑布式软件开发流程中，设计、开发、测试、编写文档等阶段是分开进行的，一个阶段必须在前一个阶段结束后才能开始。而在经验型软件开发流程中，有些阶段可以同时进行，并不需要等到前一个阶段结束再开始下一个阶段，因此开发阶段之间会有时间上的重叠。——译者注

废除了传统部门分工的概念。团队工作的速度和灵活性得到了提升，同时，共同的责任和合作还激发了成员的参与感和使命感，以及对市场的敏感度。而这种做法的代价是，你需要管理一个具有可见度、需要沟通、有紧张感甚至是冲突的流程。

- *多层次、多功能的学习*。团队中的学习是多维度的。例如，3M公司鼓励每位工程师花大概15%的工作时间来追求他们的梦想。如果一个团队在本田公司遇到了障碍，就有可能被派遣到欧洲一探究竟。也就是说，学习通常没有固定的方式和场所。最重要的是，学习源于管理者所培养和激发的员工主动性。
- *微控制*。虽然开发团队是自我管理的，但这并不意味着他们就完全不受控制了。我们的目标是既要让团队有足够的空间自我管理，又要设定足够的检查点，防止不稳定、不明确以及紧张因素带来混乱。使用同伴的无形压力以及“关爱和控制”的方法是微管理的基础。团队的这种活跃状态会使团队的隐性知识慢慢浮现，并且创造出以软件方式呈现的显性知识。这种活力只有在管理层营造的关爱氛围下才会出现。选择团队领导要小心翼翼。团队依靠引入变革及恰到好处的活力来保持其动态均衡，确保成员间能够相处融洽，协同工作。团队成员之间应该拥有相同的价值观。激励机制应该以团队为单位。团队成长过程中犯错不可避免，我们要对其有耐心。
- *知识传递*。在团队中获得的知识并不足以保证在市场上成功。来之不易的知识应该在整个公司范围内积极分享。公司依靠有经验的员工来培养新的团队，把工作中积累的项目实践提升为整个公司级的标准流程。与此同时，摒弃过去的做事方式和学习新的东西同等重要。市场日新月异，从前的方式过时了，用旧的方式做事不可能满足管理者提出的新要求。

我们发现下列要素同样能够帮助改进软件开发。

- 人。人在自我管理的时候效率最高。与其让别人替自己许下承诺，人们更重视自己许下的承诺。人在非工作时间会有很多更好的创意。人们总是尽

力把事情做到最好。然而，如果在高压下高强度工作的话，人们通常会自觉地逐渐降低工作质量。

- 团队成员。团队成员在没有外界干扰的时候效率最高。团队在自己解决问题时成长最快。面对面的深入交流是团队工作最高效的方式。
- 团队组成。团队比同等数量的个人的工作效率更高。具备所有工作必需技能的跨职能团队能够创造出更强大的产品。团队中人员的变动时常会在一定时间内降低团队的效率。

2.4　尽管我们知道该如何做

虽然我们都知道预测型或者瀑布式流程有诸多弊端，但是仍然有很多人和组织继续想方设法用这种流程获得成功。2005年，我们和玛莎百货（Marks & Spencer）[①]的首席技术官以及他的工作人员一起讨论经验型流程以及Scrum。那时候，他刚刚升级了整套开发流程，从国际咨询顾问公司普华永道（PWC）那里引进了整套的方法、工具、培训和实施服务。而当时PWC采用的流程正是预测型或者说瀑布式流程。

当时玛莎百货的首席技术官对经验型流程感到好奇，听起来似乎挺熟悉，于是想了解更多这方面的知识。随着我们向他进一步解释，他开始流露出激动的神情。他打断我们，说他的组织也在使用经验型流程。他说，一旦使用PWC流程的项目陷入困境，他们就会中断项目，然后开始使用经验型流程直到项目回到正轨，有时甚至直到项目结束。他说那是他们的“杀手锏”，也就是说他们的这一绝招能够帮助他们摆脱困境。

当我们询问使用经验型流程摆脱困境后他们会怎么办时，他却毫不顾忌地说他们会回到原来的PWC流程上。由此可见，尽管他们知道该怎么做，但并不意味着被允许这样做，除非遇上紧急情况。

① 玛莎百货是一家英国的零售商，总部位于伦敦。——译者注

2.5 敏捷性

随着我们生活的世界变得越来越复杂，商务人士和组织面临越来越多的商机。每位企业家或者梦想成为企业家的商务人士需要抓住机会，发现什么是可行的、成本会是多少以及会有哪些风险。一旦发现风险是可以接受的，企业家就会尽快付诸行动以把握机会。然而，尽管我们尽力控制风险，事情仍很容易失控。大胆和谨慎缺一不可。我们把掌控机会的能力定义为“敏捷性”，它用来衡量成功把握机会的能力。我们能够迅速灵敏地大胆采取行动，并有效地控制风险。我们能够取悦新的客户，也能让竞争对手在早晨醒来时为他们的失败而垂泪。

敏捷性是在一定风险下把握机会的能力或着说是勇于面对挑战的能力。它是现今最重要的竞争优势。我们利用了这种优势，并通过把项目控制在30天以内的方式来控制风险。

按照这种方式，我们就能够大胆尝试新的想法，不畏惧失败。我们能够在早期就知道这些想法是不是成本太高、不现实或者根本无法实现，这样就可以在投入更多金钱之前“刹车”。

2.6 小结

我们需要把握机遇并有效地响应挑战。我们也需要探索很多新点子，然后变换思路，找到最好的解决方案。如果看到机会或者想干一番事业，你不仅可以实现目标，还能够完善目标以提供最有价值的功能。拥有了更可控、更快、风险更低的流程，你就可以在30天内开发出有价值的东西并持续改进。

迭代增量式的经验型软件开发方法也已经存在20多年了。有了它，就可以在严格控制风险的同时，限时开发出软件功能增量。它在每30天（甚至更短）完成

整个业务功能增量交付的同时提供了极高的透明性，因此避免了浪费。它的敏捷性和灵活性有助于软件的调整，以适应涌现的需求，从而大大地提高了软件的适用性。我们再也不用担心软件的开发进度，再也不用担心承诺无法兑现，再也不用担心预算的增加，再也不用依赖甘特图和模型等抽象工具来分析进度，因为我们能够准确地知道至少在每30天中完成的功能和进度。