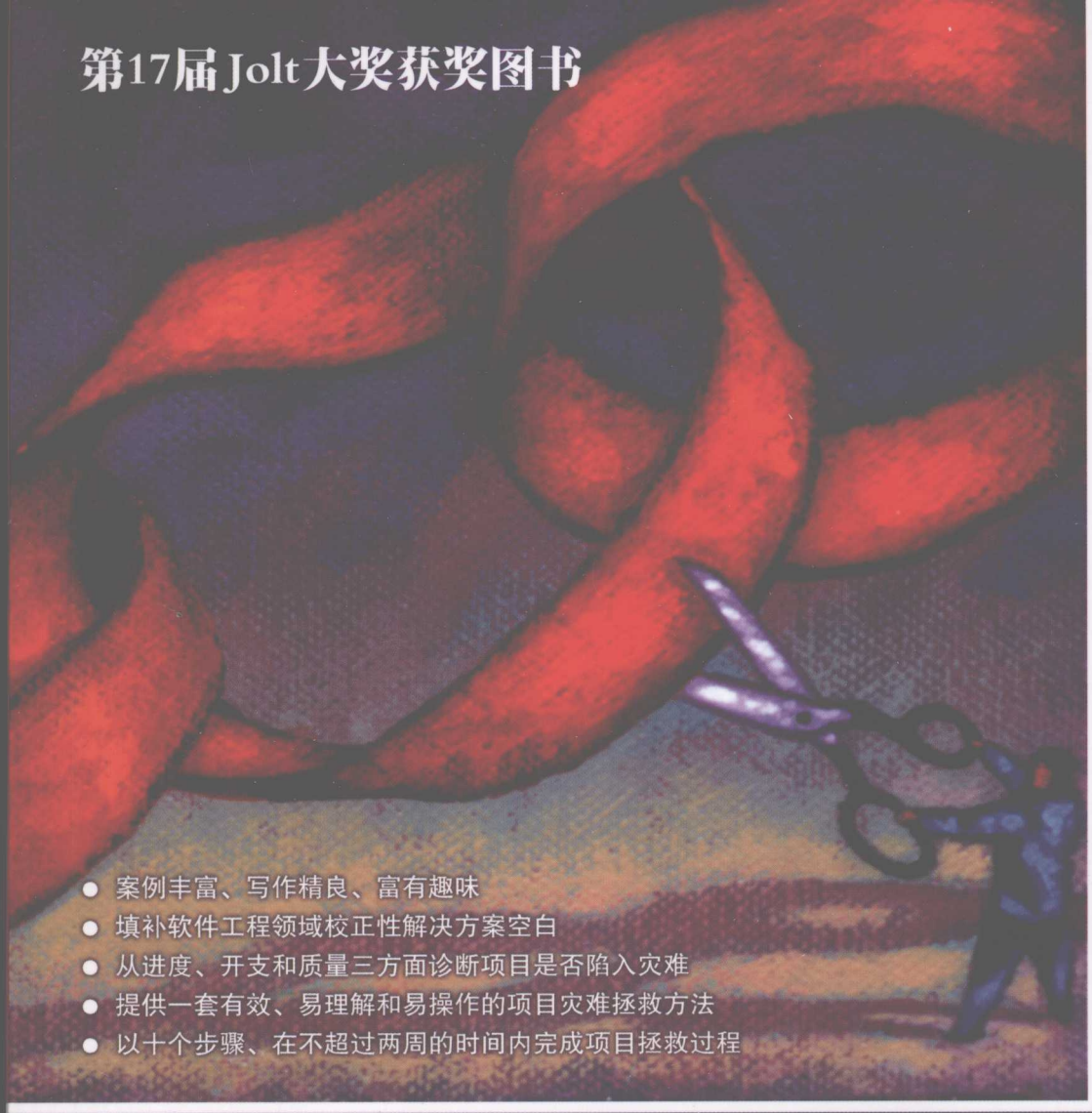
第17届Jolt大奖获奖图书



- 案例丰富、写作精良、富有趣味
- 填补软件工程领域校正性解决方案空白
- 从进度、开支和质量三方面诊断项目是否陷入灾难
- 提供一套有效、易理解和易操作的项目灾难拯救方法
- 以十个步骤、在不超过两周的时间内完成项目拯救过程

软件工程研究院

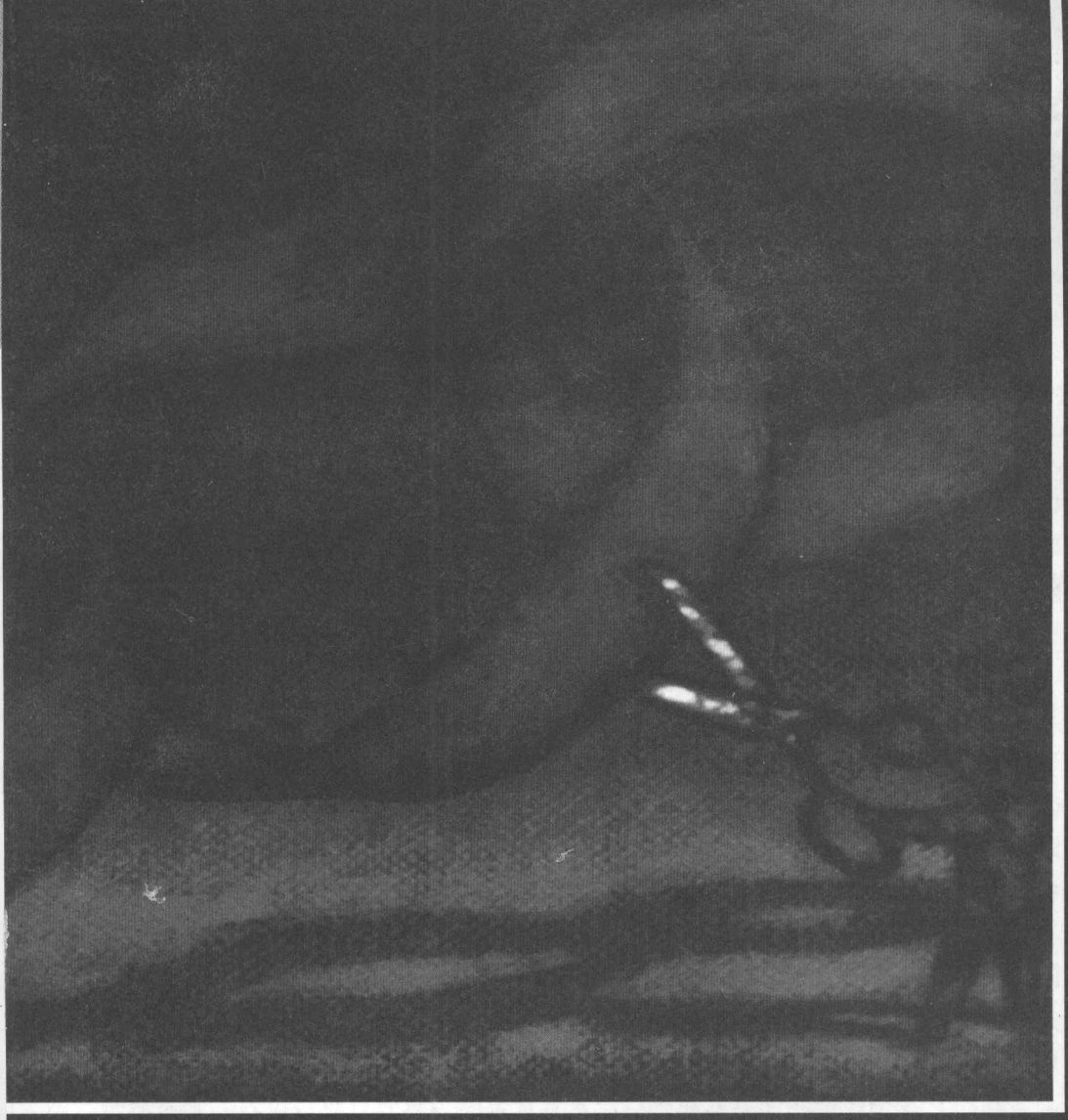
灾难拯救
——让软件项目重回轨道

Catastrophe Disentanglement

Getting Software Projects Back on Track

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

(美) E. M. Bennatan 著
侯艳飞 侯玉芳 李 萌 译
飞思科技产品研发中心 监制



软件工程研究院

灾难拯救
——让软件项目重回轨道

Catastrophe Disentanglement

Getting Software Projects Back on Track

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

(美) E. M. Bennatan 著
侯艳飞 侯玉芳 李 萌 译
飞思科技产品研发中心 监制

版　权

版权贸易合同登记号　图字：01-2007-4097

图书在版编目（CIP）数据

灾难拯救：让软件项目重回轨道 /（美）本拿坦（Bennatan,E.M.）著；侯艳飞，侯玉芳，李萌译. —北京：电子工业出版社，2008.2

（软件工程研究院）

书名原文：Catastrophe Disentanglement: Getting Software Projects Back on Track

ISBN 978-7-121-05809-7

Ⅰ. 灾… Ⅱ.①本…②侯…③侯…④李… Ⅲ.软件开发－项目管理 Ⅳ.TP311.52

中国版本图书馆 CIP 数据核字（2008）第 006118 号

责任编辑：宋兆武　张　帆

印　　刷：北京机工印刷厂

装　　订：三河市鹏成印业有限公司

出版发行：电子工业出版社

北京海淀区万寿路 173 信箱　邮编：100036

开　　本：720×1000　1/16　印张：15.75　字数：264.6 千字

印　　次：2008 年 2 月第 1 次印刷

印　　数：5 000 册　　定价：39.80 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

几年前，我听了这样一个故事。一个俄国人、一个法国人、一个日本人和一个美国人不幸被食人族抓住了。在被扔进沸水锅之前，酋长告诉他的俘虏，每人可提一个请求，而他会满足他们在这世上的最后要求。

俄国人的请求是喝最后一杯伏特加，法国人的请求是让一位年轻的本地姑娘给他最后一吻，日本人说他的请求是最后一次关于质量的演讲。最后轮到了美国人说出自己的请求，他说："请先把我扔进沸水锅吧，这样我就不必再听一次关于质量的演讲了！"

什么事都要讲个时机和场合。当一个软件项目陷入严峻的困境时，软件开发机构最想听到的是他们应怎样扭转败局，使项目进行下去。然而此时，并没有可遵循的 PMI、IEEE、SEI 或 ISO 标准帮助他们拯救项目。PMI、IEEE、SEI 和 ISO 这些组织提供了预防项目失败的方案，而没有提供拯救项目于危难的办法。当项目迫近"被扔入沸水锅"的结局之时，它最后的请求是"救我"，而不是"再告诉我一次如何避免陷入困境"。

本书是一本"救治"之书。本书探讨了如何拯救面临失败的软件项目使之回到正轨，虽然其中也偶有内容论及灾难预防。本书描述了拯救（或解救）面临失败的软件项目（或称软件项目灾难）的 10 个步骤。本书有广泛的目标读者群，包括软件开发人员、项目经理、高级管理人员以及软件项目利益攸关者（同软件项目之间存在很大利益关系的人）。

本书也旨在成为一本教材。在本书每一章的末尾，都有本章小结，并提供了习题。

本书中有的内容要求读者具备一些软件工程知识，不过这样的内容很少，具备软件工程知识并不是使用本书的必要条件。本书对于缺少管理知识的软件工程师和对于缺少软件开发知识的项目管理人员一样有用。

本书共包括 13 章：

- 第 1 章为绪论。本章介绍了灾难拯救的概念，探讨了软件项目何时需要拯救行动的介入。在本章中，还对本书中使用的几个基本术语进行了阐释。

- 第 2 章论述了判断项目灾难来临的方法。对于遇到麻烦的项目来说，本章是决定是否需要对它采取后面各章中描述的拯救步骤的一章。
- 第 3 章至第 12 章描述了灾难拯救过程的 10 个步骤(每章描述一个步骤)。
- 第 13 章为结束语，标题是“把最后的拼图放入位置”。在本章中，有些内容是关于灾难预防的。本章综览之前各章描述的 10 个步骤，并阐述了如何进行时间安排以使全部拯救过程在两周时间里完成。

本书所介绍的拯救过程中，很多步骤之间存在交叠，且每一步骤都依赖于它前面的步骤，因此，本书不适合随兴翻到哪里看哪里、“东一榔头西一棒槌”的阅读方式。如果读者从头到尾地了解了整个拯救过程，那么，理解起每个拯救步骤来也会更加容易。因此，强烈建议您在实施拯救过程之前从头到尾学习本书全部内容。不过，您也不必因为我的上述建议而气馁，本书每一章都有“本章小结”，您可以采用仅仔细阅读与正在实施的拯救步骤有关的章而对其他各章只阅读“本章小结”的简化方式来学习。

本书内容偏重实践而非理论。本书在介绍很多方法和技术时，并未说明其理论基础。不过，本书中提供了大量的参考书目，以飨对理论背景感兴趣的读者。参考书目信息见本书末尾部分。

在本书出版之前，有一些关于项目灾难局面能否扭转的讨论，实际上也就是说，当我们想拯救一个项目的时候，将这个项目称为灾难是否合适。对于任何一名在大型技术公司工作过并听到过某位沮丧的高级经理说“这个项目是个灾难!”的人来说，答案都是很明显的。如果灾难局面不可扭转，那么，项目在彼时彼地就会被放弃了，但是，高级经理接下来通常会说的是:“我们需要马上使它回到轨道!”是的，这正是本书所要论述的。

我在摩托罗拉公司和其他一些技术公司从事了多年的软件项目管理工作，并对数百家软件开发机构的软件项目开发数据进行了收集和分析，这是形成本书中“拯救”概念的基础。在本书之前，我曾写过一篇同名的论文，这篇论文发表在了美国国防部出版的期刊“*CrossTalk, The Journal of Defense Software Engineering*”上。

感谢埃米尔（Amir）在本书问世过程中给予的极大帮助，他的贡献和评论是无价的。

E. M. Bennatan

2006 年 1 月

反侵权盗版声明

目 录

第 1 章 绪论……1
1.1 灾难拯救过程概述……4
1.1.1 案例研究……4
1.1.2 做出拯救决定……5
1.1.3 拯救过程……6
1.2 一些调查数据……8
1.3 一些提示……10
1.4 本章小结……13
第 2 章 确定项目是否陷入灾难……15
2.1 进度……18
2.1.1 设置进度警报器……19
2.1.2 调整进度警报器……22
2.1.3 监视延长后的时间表……23
2.2 预算……26
2.2.1 设置预算警报器……27
2.2.2 其他需考虑的事项……31
2.3 质量……31
2.3.1 问题列表警报器……32
2.3.2 顾客满意度警报器……35
2.4 学会利用经验……36
2.5 本章小结……37
习 题……39
第 3 章 第 1 步——停止……43
3.1 停止项目……44
3.1.1 为什么停止项目……44
3.1.2 谁来停止项目……45
3.1.3 项目停止程序……45

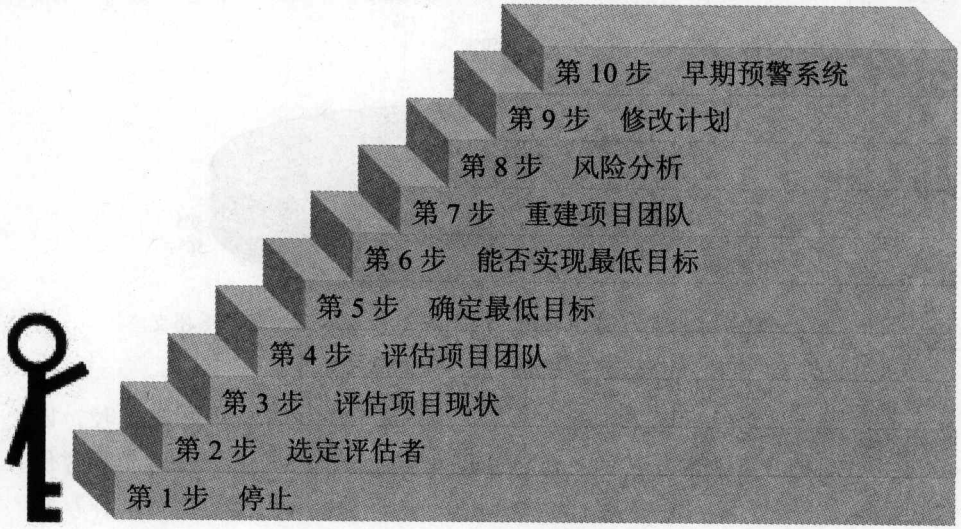
3.2 准备下一步……47
3.3 开展团队行动……48
3.4 处理反对意见……50
3.5 可能出现哪些问题及如何解决……51
3.6 本章小结……53
习 题……54
第 4 章 第 2 步——选定评估者……57
4.1 该选谁——合格评估者的素质要求……60
4.2 案情陈述……61
4.2.1 应包含的内容……61
4.2.2 管理者的承诺……63
4.2.3 评估者的承诺……65
4.3 大型软件项目……66
4.4 可能出现哪些问题及如何解决……68
4.5 本章小结……70
习 题……71
第 5 章 第 3 步——评估项目现状……73
5.1 评审……76
5.1.1 软件项目评审概述……76
5.1.2 评审面临失败的软件项目……78
5.2 项目状态信息的来源……80
5.2.1 口头的状态信息……81
5.2.2 操作性的状态信息……82
5.2.3 文档类信息……83
5.3 评估大型软件项目……83
5.3.1 大型项目有何不同……84
5.3.2 评估团队……85
5.3.3 评估大型项目的指导方针……86
5.4 拼拼图……87
5.5 可能出现哪些问题及如何解决……89
5.6 本章小结……91
习 题……92

第 6 章　第 4 步——评估项目团队 ····· **95**
6.1　一般原则 ····· 97
6.2　评审团队整体 ····· 99
6.3　评审项目管理 ····· 102
6.4　评审团队成员 ····· 104
6.5　整合信息 ····· 105
6.6　可能出现哪些问题及如何解决 ····· 106
6.7　本章小结 ····· 108
习　题 ····· 109
第 7 章　第 5 步——确定最低目标 ····· **111**
7.1　项目目标和拯救过程 ····· 113
7.1.1　区分目标、具体目标、需求和交付成果 ····· 113
7.1.2　项目目标由谁制定 ····· 115
7.1.3　同目标监督者结成同盟 ····· 116
7.2　目标最低化的准则 ····· 117
7.2.1　降低目标的过程 ····· 118
7.2.2　一个降低目标的案例 ····· 119
7.2.3　处理反对意见 ····· 121
7.3　大型项目的目标最低化 ····· 122
7.4　可能出现哪些问题及如何解决 ····· 123
7.5　本章小结 ····· 127
习　题 ····· 128
第 8 章　第 6 步——确定最低目标能否实现 ····· **131**
8.1　可实现的目标 ····· 132
8.1.1　可行性分析方法 ····· 133
8.1.2　被拯救项目的可实现目标 ····· 134
8.1.3　如果目标不可实现 ····· 136
8.2　中期报告 ····· 137
8.3　可能出现哪些问题及如何解决 ····· 138
8.4　本章小结 ····· 140
习　题 ····· 141

第 9 章　第 7 步——重建项目团队 ····· 143
9.1　回顾团队评估 ····· 145
9.2　识别问题 ····· 146
9.3　重建团队 ····· 149
9.3.1　应对变更 ····· 150
9.3.2　实施变更 ····· 151
9.3.3　处理反对意见 ····· 153
9.4　重建大型项目团队 ····· 155
9.5　可能出现哪些问题及如何解决 ····· 157
9.6　本章小结 ····· 158
习　题 ····· 160
第 10 章　第 8 步——风险分析 ····· 161
10.1　风险分析概述 ····· 163
10.2　风险分析过程 ····· 165
10.2.1　预见问题 ····· 165
10.2.2　分析阶段 ····· 167
10.2.3　实施风险行动方案 ····· 169
10.3　风险分析的一个例子 ····· 170
10.4　可能出现哪些问题及如何解决它 ····· 172
10.5　本章小结 ····· 174
习　题 ····· 176
第 11 章　第 9 步——修改计划 ····· 177
11.1　软件项目计划制定综述 ····· 178
11.1.1　软件项目计划概念 ····· 179
11.1.2　软件项目开发计划 ····· 180
11.1.3　项目计划制定工具 ····· 183
11.2　一个被拯救项目的计划制定 ····· 184
11.2.1　制定拯救项目计划指南 ····· 185
11.2.2　额外需要考虑的事项 ····· 188
11.3　可能出现哪些问题及如何解决 ····· 190
11.4　本章小结 ····· 193
习　题 ····· 194

第 12 章　第 10 步——创建早期预警系统 ……197
12.1　早期预警系统的组成要素 ……200
12.2　开发数据收集 ……204
12.2.1　项目开发数据的作用 ……204
12.2.2　重启后项目的数据收集 ……207
12.3　定期项目现状评审 ……209
12.4　项目报警机制 ……211
12.5　启动校正行动 ……213
12.6　后续行动 ……214
12.7　可能出现哪些问题及如何解决 ……215
12.8　本章小结 ……217
习　题 ……218
第 13 章　尾声：把最后的拼图放入位置 ……221
13.1　项目结束后的总结回顾 ……222
13.2　人为因素 ……225
13.3　灾难拯救的时间表 ……226
13.4　最终报告 ……227
13.5　案例分析 ……228
13.6　结束语 ……229
参考书目 ……231
术语表 ……240
关于作者 ……241

第1章 绪 论



在斯宾塞·约翰逊（Spencer Johnson）所著的《谁动了我的奶酪》（*Who Moved My Cheese*）一书中，即使原先有奶酪的地方已经没有了奶酪，小矮人们还是一如既往地到那里去找奶酪。在旁观者看来墨守陈规已经没有意义，但是，继续以以前的行为方式行动却是一种常见的当局者行为。当软件项目陷入困境之时，当局者墨守陈规的现象相当普遍。我们步履艰难地重复以前的行为，期望问题会消失，“奶酪”会奇迹般地再出现。然而，在数不胜数的这类案例中，结果都是期望落空。

就像缠线团一样，当线缠乱了纠绕在一起的时候，明智之举是先停下来，否则只会越缠越乱。对于陷入灾难的项目来说，同样是这个

道理。我们继续以前行为的时间越长，情况就会变得越糟糕；正确的举措应是暂停所有的行动并重新评估我们正在做的事情。

软件项目灾难是指软件项目在进度、预算、质量这三个方面中的一个或多个方面完全失控的情况。软件项目灾难并不罕见，在被调查的软件开发机构中，44%的公司称他们由于严重超支问题撤销或放弃过软件项目，15%的公司说他们有超过10%的软件项目由于严重超支问题而遭放弃（见图1.1）。

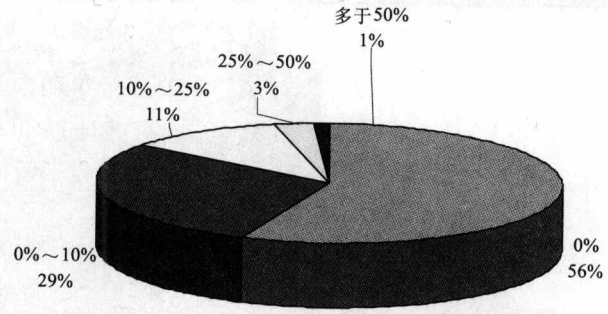


图1.1　被调查的机构在过去三年中由于成本或时间严重超支而放弃或撤销了的软件项目的比例[12]

但是显然，并不是每一个超支或质量问题都意味着项目失控，因此，我们应于何时判定一个软件项目陷入了灾难？采取果断步骤暂停以前所有行动的准则是什么？我们如何着手项目的重新评估工作？最重要的，我们如何着手使项目再次前进？这些问题的答案是灾难拯救概念的要义。

不久前结束了对丹佛机场行李处理系统进行的一次拯救尝试，该项目耗资数亿美元，自十多年前就陷入了灾难，十多年间已对其进行过数次拯救尝试①。2005年8月，这一声名狼藉的系统终于被弃用，这一幕让人想起了库布里克那部令人难忘的太空历险影片中哈尔的终

①丹佛机场行李处理系统项目最初预算近2亿美元，原定于1994年4月完成，但延期了几个月才完成，这导致丹佛国际机场的开张推迟到了1995年2月。据报道，延期开张给丹佛机场带来了大约每天100万美元的损失。这一自动化行李处理系统投入使用之后，一直受机械和软件问题的困扰而未能很好地工作，行李误装、误转或掉出传送车引起系统混乱的情况时有发生。——译者注

结[①]。这个项目由于每延期一天会带来 100 万美元的损失而恶名远扬。关于丹佛机场行李处理系统项目的一个有意思的问题是：为什么一次次的拯救努力都没有取得成功？

在困扰这一项目的所有问题中[3,4]，最令人生畏的可能是项目的难以实现的目标。由于该项目所提出的超前的系统功能已成为该项目的主要魅力所在，任何与该项目有关的人都不可能使项目目标发生重大改变。然而，任何灾难拯救过程的基础都是能够界定出可实现的目标。无法界定出可实现的目标，灾难拯救过程就不可能取得成功，这是丹佛机场行李处理系统拯救失败的一个主要原因。

正如前面的调查数据所示，像丹佛机场行李处理系统项目这样的项目灾难案例并不少见（虽然鲜有像丹佛机场行李处理系统项目这样极端的）。大部分软件开发机构无须看这些调查数据就都明白这一点。加拿大财政委员会的马丁·科布（Martin Cobb）有一段很有名的话，很好地表达了这一令人沮丧的现实："我们知道项目为什么失败，我们知道如何预防它们失败，那么为何它们还是失败？"[10]

科布的话强调了软件工程的常规方法。已有的软件工程策略的目标是预防软件灾难的发生，即预防项目失控。在软件工程中，策略有着重要的作用。然而，五十多年的历史表明，尽管有这样那样预防项目失控的方法，软件灾难还是会继续存在，短期内不会消失。

在软件项目失控之时，并没有可遵循的 PMI、IEEE、SEI 或 ISO 标准帮助当事人对项目进行拯救。PMI、IEEE、SEI 和 ISO 这些组织提供了预防项目失败的方案，而没有提供拯救项目于危难的办法。然而，失控的项目必然要失败吗？本书下面的内容将向你证明失控的项目决非只有失败一种命运。

本书填补了软件工程领域矫正性解决方案方面的空白。本书的研究对象是已陷入严峻困境的项目，本书不会过多关注我们是如何陷入困境的，而是更关注我们如何摆脱困境。

①哈尔（Hal）是斯坦利·库布里克（Stanley Kubrick）的影片《2001：太空奥德赛》（*2001: A Space Odyssey*）中那台违背人类命令的电脑。

1.1 灾难拯救过程概述

在实施拯救过程的第一个步骤之前，我们必须先确定整个过程是必需的，也就是做出这样的判定：如果项目按照目前的方式继续做下去而不采取任何激烈的改变措施，那么它将基本上没有机会成功。

很多软件机构难以做出这样的判定，有些软件机构则完全回避做这样的判定。事实上，在采取适当的拯救行动之前，陷入困境的项目以原先的方式继续执行的时间太长，是一个很普遍的现象[6]。科尔（Keil）[7]使用“脱缰者”（runaways）一词来形容无法达成其目标却继续占用有价值资源的项目。科尔所说的“脱缰者”实际上就是陷入灾难却长久未进行灾难诊断的项目。确实，项目能否挽救通常取决于对项目灾难进行诊断的早晚，而且，允许“脱缰”的项目以原先的方式继续执行的公司是在浪费有价值的资源。下面的案例是上述事实的一个很好的例证。

1.1.1 案例研究

下文所描述的“FINALIST”案例例证了这一现实：尽管项目存在的问题在几乎任何一个旁观者看来已很明显，当局者认识到项目陷入严峻困境还是相当困难的。这个案例并不独特，但很具有代表性，因而值得关注。它说明了在通向失败的道路上越走越远是何等容易。

千禧年过去了，软件将遭遇世界末日的预言成为过眼云烟，这时，一家加拿大软件机构发现它的一个小商业部件几乎没有了客户。该部件的主要专长在于支持 Cobol 程序（Cobol 程序被认为是很多千年虫的藏身之地），而突然之间没有了足够的 Cobol 工作来支援它。

为此，这家公司决定重写其核心产品之一的 FINALIST。FINALIST 是一个大型的金融分析系统，为了保持公司在解决千年虫问题（公司认为千年虫还会出现）上的独特专长，公司选择了以 Cobol 编程语言来重写该系统。新项目被命名为了 FINALIST2，计划用 30 个月的时间，由 14 名开发人员组成的一个团队（其中 8 名是经验丰富的 Cobol 程序员）来完成。

转眼之间项目进入了第二个年头。这一年刚开始，就有两名 Cobol 程序员退了休，之后不久，又有三名程序员跳槽到了别的公司。在只剩下三名经验丰富的

Cobol 程序员的情况下，FINALIST2 项目开始经历严重的问题和进度滞后。公司的管理层再三驳回了重新评估项目的请求，并试图通过经常做总结、增加人手、提供激励、延长项目时间等措施来使项目回到轨道。

在项目开始后的第 28 个月上，才终于引入了一位顾问。这位顾问的第一个建议就是立即将项目暂停。顾问开这一猛药是基于这一结论的：该项目目前基本上没有进展或者说无法取得有意义的进展，而且该项目也不可能按预定计划完成。周围没有足够的可参与该项目的、有经验的 Cobol 程序员，同时，不可能新招聘到有经验的 Cobol 程序员，而且，新手也不可能在合理的时间范围内变得十分精通 Cobol。

最后的建议是使用现代编程语言重新启动该项目，或者干脆撤销该项目。

在这个案例中，关键点之一在于管理层没有注意到曾经的优势（Cobol）已经不再是优势了——一个典型的“谁动了我的奶酪”的例子。保护公司的 Cobol 专长这一强烈愿望显然是这次失败的孕育之壤，而对承认错误的本能抵触（反对对该项目进行重新评估）也导致了这次失败。这两个因素使其难以产生有成效的解决方案。管理层努力调整了项目的几乎每个方面（过程、团队、时间计划），但是就是没有去解决问题本身。

这个案例说明了决策者对采取激烈手段的必要性存在着接受上的难度，它让人联想到不能起身离开赌桌的赌徒。首先，期望着传统的方法最终能使项目重回轨道，而拖延着不做艰难的决定是一种本能的行为趋向。其次，还有一个难题就是对以前的决定负责，这往往会促使决策者投入更多的资源以避免承认错误（我们已经知道，这只会使困境加深[6]）。

项目陷入困境决不是出其不意从天而降的事情，而且，那些在经历失败的道路上前进的人往往知道项目中有的地方很不对劲。然而，“很不对劲”有多严重？我们如何知道何时该采取激烈手段了？理想情况下，应该有一个决策规则系统（一种检测软件），管理者可以利用它来监控他们的项目，而且，它还可以替他们做出决定。

1.1.2　做出拯救决定

现实中没有完美的灾难检测器。然而，尽管对一个项目做出完全客观的决定

是困难的，还是有很多在很大程度上排除了决策中的主观因素的方法。这些方法要求对项目进行深入的评估以及付出相当大的努力。与现状报告或者定期的进展总结不同，这些方法并不是设计来在软件开发周期全过程中每间隔一定时间使用一次，而是只应用在我们怀疑项目可能陷入了严峻困境但是还并不确定是否需要通过“手术”来挽救它的时刻。

诊断程序基于对项目三个基本方面的评估：

- 进度
- 预算
- 质量

在诊断过程中，检查项目的上述方面是否存在着严重的问题，情况是否正在恶化而非好转。上述三个方面中的任何一个方面都可能引发出“项目已陷入灾难”这一判定结果，不过，灾难发生时，常常是三个方面都存在着严重的问题。第 2 章“确定项目是否陷入灾难”就这个话题做了详细论述，还探讨了质量好坏（对质量好坏的界定是以产品的缺陷水平和顾客或用户对产品的满意程度为基础的）这个棘手的问题。

一旦做出了项目确实已陷入灾难的决定，那么选项就变得更为清楚：要么拯救，要么放弃。如果选择拯救，那么，便到了本书所论述的 10 个步骤的拯救过程该出场的时间了。

1.1.3 拯救过程

本书所设计的拯救过程旨在拯救陷入严峻困境的项目，通过实施这一拯救过程，能够确定出对项目进行事务性或战略性调整的方案。拯救过程围绕着两个重要人物建立：发起人（发起拯救过程，并监督拯救过程的实施）和项目评估者（领导并实施拯救过程）。发起人是一位局中人，是软件开发机构中对当前项目负责的一位高级经理；项目评估者是一位局外人，是一位经验丰富的、可信赖的、不偏不倚的专业人士。

灾难拯救过程包括下述 10 个步骤。

第 1 步　停止：暂停所有的项目开发活动，并给项目团队分配支持拯救工作的任务。