

PEARSON

40周年中文
纪念版

THE
Mythical
Man—Month
Essays on Software Engineering, Anniversary Edition

人月 神话

(Frederick P. Brooks, Jr.)

[美]小弗雷德里克·布鲁克斯/著

UMLChina翻译组 汪颖/译

PEARSON

ALWAYS LEARNING ALWAYS LEARNING ALWAYS LEARNING

清华大学出版社

人 月 神 话

(40 周年中文纪念版)

[美] 小弗雷德里克·布鲁克斯 著
UMLChina 翻译组 汪颖 译

清华大学出版社
北 京

Authorized translation from the English language edition, entitled The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition, 9780201835953 by Frederick P. Brooks, JR., published by Pearson Education, Inc, publishing as Addison-Wesley, Copyright © 1995.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

CHINESE SIMPLIFIED language edition published by PEARSON EDUCATION ASIA LTD., and TSINGHUA UNIVERSITY PRESS Copyright © 2015.

本书中文简体翻译版由培生教育出版集团授权给清华大学出版社出版发行。未经许可,不得以任何方式复制或抄袭本书的任何部分。

北京市版权局著作权合同登记号 图字: 01-2002-5152

本书封面贴有 Pearson Education(培生教育出版集团)防伪标签,无标签者不得销售。

版权所有,侵权必究。侵权举报电话: 010-62782989 13701121933

图书在版编目(CIP)数据

人月神话(40周年中文纪念版)/(美)布鲁克斯(Brooks, F. P.) 著; UML China 翻译组, 汪颖 译. —北京: 清华大学出版社, 2015(2015.6重印)

书名原文: The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition
ISBN 978-7-302-39264-4

I. ①人… II. ①布… ②U… ③汪… III. ①软件工程 IV. ①TP311.5

中国版本图书馆 CIP 数据核字(2015)第 024400 号

责任编辑: 陈莉 高 岫

封面设计: 周晓亮

版式设计: 牛静敏

责任校对: 曹 阳

责任印制: 杨 艳

出版发行: 清华大学出版社

网 址: <http://www.tup.com.cn>, <http://www.wqbook.com>

地 址: 北京清华大学学研大厦 A 座 邮 编: 100084

社 总 机: 010-62770175 邮 购: 010-62786544

投稿与读者服务: 010-62776969, c-service@tup.tsinghua.edu.cn

质 量 反 馈: 010-62772015, zhiliang@tup.tsinghua.edu.cn

印 刷 者: 清华大学印刷厂

装 订 者: 北京市密云县京文制本装订厂

经 销: 全国新华书店

开 本: 180mm×250mm 印 张: 24.5 字 数: 316 千字

版 次: 2015 年 4 月第 1 版 印 次: 2015 年 6 月第 3 次印刷

印 数: 10001~13000

定 价: 68.00 元



神品——代序*

这是本书中唯一的一节废话。

我是一个书狂，积习甚深，费尽心机在软件工程、系统工程方面积累了一些书。书，在我看来当分为神品、精品和普通三等，其中神品、精品又分别有一、二和三品之分。我所收集的书中，软件工程书大都属于精品，神品只有两本，Frederick P. Brooks 的这本书就属于神品之列。

软件作为一个行业，逐步背起了“solving the wrong problem” (解决错误的问题)的名声。问题决定解决方案，这也就是说，我们一直在制造错误解决方案！这方面有大量的证据，其中最著名的是美国政府统计署(GAO)的数据：全球最大的软件消费商(美国军方)每年要花费数十亿美元购买软件，而在其所购软件中，可直接使用的只占 2%，另外 3%需要做一些修改，其余 95%都成了垃圾。一句话，不管这些软件是否符合需求规格，它们都显然没有满足客户的需求。面向对象技术并没有给我们带来“神奇的效应”，不管开发商如何吹嘘面向对象 OO(Object-Oriented)工具多么万能，也不管那些 OO 狂热者多么毅然地前赴后继，这方面的数据从 20 世纪 80 年代以来并没有发生大的改观。

这实在令我们的软件工程专家和从业人们羞愧，因为它揭示了我

* 编注：该序的作者是王计斌(Dave Wang)，清华大学博士，研究方向包括软件工程和集成产品开发(IPD)，长期从事 IPD/CMM 推行，创办软件工程研究与实践论坛(<http://www.seforum.net>)，现在华为技术有限公司工作。

们可能一开始就从根本上做错了什么！20 世纪 90 年代中期，当软件工程一代宗师 Michael Jackson(非歌坛巨星 Michael Jackson)宣布他们的研究结果时，立刻在软件工程界激起了阵阵涟漪。Jackson 指出，软件从业人员和方法学大师们只是简单地模仿和照搬其他学科的方法，却将最重要的方面(问题域)忽略了。他指出，面向对象方法和结构化方法对问题域的处理没有什么大的区别，却被人们过分地用美好的词汇美化了：

“...You can see the results clearly in many object-oriented modeling descriptions. Often they are accompanied by fine words about modeling the real world. But when you look closely you can see that they are really descriptions of programming objects, pure and simple. Any similarity to real-world objects, living or dead, is purely coincidental...”

(……从众多面向对象建模的描述中，你可以很清楚地看到这些恶果。而且它们还经常伴随着有关现实世界建模的非常美好的词汇。然而，仔细看看，你就会发现它们其实是彻头彻尾的编程对象！如果说有任何和现实世界对象相似的地方，不管是死是活，纯属巧合……)

回首软件工程近 40 年的发展，Jackson 哀叹软件行业普遍缺乏专业性，充满了业余人员，“手中有一个锤子，看到什么都是钉子”，谁都可以开发性命攸关的软件。

这就是我们面临的严峻而复杂的现实，也许您会感到震惊！然而在大师 Frederick P. Brooks 眼里，是那么的平静。因为早在 28 年前，他就在《人月神话》(*The Mythical Man-Month*)这本不朽著作中对这些内容做了深入论述。

这本小册子行文优美，思想博大精深，字字真言，精读之有不尽的趣味，藏之又是极珍贵的文献，名眼高人，自能鉴之。



前些年，一位朋友从印度归来，说此书在印度极为普及，我也动起笔来，但惭愧终未成正果。汪颖兄素来勤恳，明知此翻译为“success without applause, diligence without reward”（没有掌声的成功，没有回报的勤勉），却兢兢业业，反复琢磨，历经单调、繁琐、艰辛的劳动，终于付梓。钦佩之余随即作序共勉。

Dave Wang

SE Forum China

2002 年 3 月于深圳



20 周年纪念版序言

令我惊奇和高兴的是,《人月神话》在 20 年后仍然继续流行,印数超过了 250 000 册。人们经常问,我在 1975 年提出的观点和建议,哪些是我仍然坚持的,哪些是已经改变了的,又是怎样改变的?尽管我在一些讲座上也分析过这个问题,但我还是一直想把它写成文章。

Peter Gordon 现在是 Addison-Wesley 的出版伙伴,他从 1980 年开始和我共事。他非常有耐心,对我帮助很大。他建议我们准备一个纪念版本。我们决定不对原版本做任何修订,只是原封不动地重印(除了一些无足轻重的修正),并用更新的思想来扩充它。

第 16 章重印了一篇在 1986 年 IFIPS 会议上的论文“没有银弹:软件工程的根本和次要问题”(No Silver Bullet: Essence and Accidents of Software Engineering)。这篇文章来自我在国防科学委员会主持军用软件方面研究时的经验。我当时的研究合作者,也是我的执行秘书,Robert L. Patrick,他帮助我回想和感受那些做过的软件大项目。1987 年,IEEE 的《计算机》(Computer)杂志重印了这篇论文,使它传播得更广了。

“没有银弹”被证明是富有煽动性的,它预言 10 年内没有任何编程技巧能够给软件的生产率带来数量级上的提高。10 年只剩下一年了,我的预言看来是安全了。“没有银弹”激起了越来越多文字上的剧烈争论,比《人月神话》还要多。因此在第 17 章,我对一些公开的批评做了说明,并更新了在 1986 年提出的观点。

在准备《人月神话》的回顾和更新时，一直在进行的软件工程研究和经验已经批评、证实或否定了少数书中断言的观点，也影响了我。剥去辅助的争论和数据后，把那些观点粗略地分类，对我来说是很有帮助的。我在第 18 章列出了这些观点的概要，希望这些单调的陈述能够引来争论和证据，然后得到证实、否定、更新或精炼。

第 19 章是一篇更新的短文。读者应该注意的是，新观点并不像原来的书一样来自我的亲身经历。我在大学里工作，而不是在工业界，做的是小规模的项目，而不是大项目。自 1986 年以来，我就只是教授软件工程，不再做这方面的研究。我现在的研究领域是虚拟环境及其应用。

在这次回顾的准备过程中，我找了一些正在软件工程领域工作的朋友，征求他们现在的观点。他们很乐意与我分享他们的想法，并仔细地对草稿提出了意见，这些都使我重新受到启发。感谢 Barry Boehm、Ken Brooks、Dick Case、James Coggins、Tom DeMarco、Jim McCarthy、David Parnas、Earl Wheeler 和 Edward Yourdon。感谢 Fay Ward 对新的章节进行了出色的技术加工。

感谢我在国防科学委员会军事软件工作组的同事 Gordon Bell、Bruce Buchanan、Rick Hayes-Roth，特别是 David Parnas，感谢他们的洞察力和生动的想法。感谢 Rebekah Bierly 对第 16 章的论文进行了技术加工。我把软件问题分成“根本的”和“次要的”，这是受 Nancy Greenwood Brooks 的启发，她在一篇“Suzuki 小提琴教育”的论文中应用了这样的分析方法。

在 1975 年版本的序言中，Addison-Wesley 出版社按规定不允许我在书中向该社的一些扮演了关键角色的员工致谢。可是，有两个人员的贡献必须特别指出：执行编辑 Norman Stanton 和美术指导 Herbert Boes。Boes 设计了优雅风格的版式和封面，他在评注时特别提到：“页边的空白要宽，字体和版面要有想象力。”更重要的是，他提出了至关重要的建议：

为每一章的开头配一幅图片(当时我只有“焦油坑”和“兰斯大教堂”的图片)。寻找这些图片使我多花了一年的时间，但我永远感激这个忠告。

Soli Deo Gloria——愿神独得荣耀。

Frederick P. Brooks, Jr.
Chapel Hill, N.C.
1995 年 3 月



第 1 版序言

在很多方面，管理一个大型的计算机编程项目与管理其他行业的大型工程很相似——比大多数程序员所认为的还要相似；在另外一些方面，它又有差别——比大多数职业经理人所认为的差别还要大。

这个领域的知识在于累积。现在，AFIPS(美国信息处理学会联合会)已经有了一些讨论和会议，也出版了一些书籍和论文，但是还没有成形的办法对这一领域来进行系统地阐述。提供这样一本主要反映个人观点的小书看来是合适的。

虽然我原来从事计算机科学的编程方面的工作，但是在 1956—1963 年，自动控制程序和高级语言编译器开发出来的时候，我主要参加的是硬件构架方面的工作。1964 年，我成为操作系统 OS/360 的经理，我发现前些年的进展使编程世界改变了很多。

虽然是失败的，但管理 OS/360 的开发仍是一次很有帮助的经历。负责这次开发项目的团队，包括我的继任经理 F. M. Trapnell，有很多值得自豪的东西。该系统在设计和执行方面都很出色，并被成功地应用到很多领域，特别是设备独立的输入输出和外部库管理，在很多技术革新中被广泛复制。现在，这一系统是十分可靠的，相当有效且非常通用。

但是，并不是所有的努力都是成功的。所有 OS/360 的用户很快就能发现它应该能够做得更好。设计和执行上的缺陷在控制程序中特别普

遍,相比之下,语言编译器就好得多。大多数缺陷发生在1964—1965年的设计阶段,所以这肯定是我的责任。此外,这个产品发布推迟了,需要的内存比计划中的要多,成本也是估计的好几倍,而且第一次发布时并不能很好地运行,直到发布了几次以后,问题才得以解决。

按照当初接受OS/360任务时的协议,在1965年离开IBM后,我来到Chapel Hill。我开始分析OS/360的经验,看能不能从中学到什么管理和技术上的教训。特别要说明的是,System/360硬件开发和OS/360软件开发中的管理经验是大相径庭的。对Tom Watson关于为什么编程难以管理的探索性问题,这本书是一份迟来的答案。

在这次探索中,我和1964—1965年的经理助理R. P. Case,还有1965—1968年的经理F. M. Trapnell进行了长谈,从中受益很多。我还对比了其他大型编程项目经理的结论,这些项目经理包括M. I. T.的F. J. Corbato,贝尔电话实验室的V. Vyssotsky和John Harr,International Computers Limited的Charles Portman,苏联科学院西伯利亚分部计算实验室的A. P. Ershov和IBM的A. M. Pietrasanta。

我自己的结论体现在下面的文字中,送给专业程序员、职业经理,特别是程序员的职业经理。

虽然写出来的是各自独立的章节,但本书还是有一个中心的论点,特别包含在第2~7章。简言之,我相信由于人员的分工,大型编程项目碰到的管理问题和小项目碰到的管理问题区别很大;我相信关键需要的是维持产品自身的概念完整性。这几章探讨了其中的困难和解决的方法。而后续的章节则探讨了软件工程管理的其他方面。

这个领域的文献并不多,但散布很广。因此我尝试在书后给出了参考文献,说明某个特定知识点并指导感兴趣的读者去参阅其他有用的文献。很多朋友读过了本书的手稿,其中一些朋友还给出了很有帮助的意

见。这些意见很有价值，但为了不打乱文字的通顺，我把它们作为注解包含在本书中。

因为这本书是一部文集而不是一部教材，所有的参考文献和注解都被放到书的末尾，建议读者在读第一遍时略去不看。

深深地感谢 Sara Elizabeth Moore 小姐、David Wagner 先生和 Rebecca Burris 夫人，他们帮助我准备了手稿。感谢 Joseph C. Sloane 教授在图解方面的建议。

Frederick P. Brooks, Jr.

Chapel Hill, N.C.

1974 年 10 月



目 录

第 1 章 焦油坑	1
编程系统产品	4
职业的乐趣	6
职业的苦恼	8
第 2 章 人月神话	11
乐观主义	14
人月	16
系统测试	19
空泛的估算	21
重复产生的进度灾难	22
第 3 章 外科手术队伍	27
问题	30
Mills 的建议	32
如何运作	35
团队的扩建	36
第 4 章 贵族专制、民主政治和系统设计	39
概念的完整性	42

获得概念的完整性	43
贵族专制统治和民主政治	44
在等待时, 实现人员应该做什么	47
第 5 章 画蛇添足	51
结构师的交互准则和机制	54
自律——开发第二个系统所带来的后果	55
第 6 章 贯彻执行	59
文档化的规格说明——手册	62
形式化定义	63
直接整合	66
会议和大会	66
多重实现	68
电话日志	68
产品测试	69
第 7 章 为什么巴比伦塔会失败	71
巴比伦塔的管理教训	75
大型编程项目中的交流	76
项目工作手册	76
大型编程项目的组织架构	80
第 8 章 胸有成竹	85
Portman 的数据	89
Aron 的数据	90
Harr 的数据	90
OS/360 的数据	92

Corbató的数据	93
第 9 章 削足适履	95
作为成本的程序空间	98
规模控制	99
空间技能	100
数据的表现形式是编程的根本	102
第 10 章 提纲挈领	105
计算机产品的文档	108
大学科系的文档	110
软件项目的文档	110
为什么要有正式的文档	111
第 11 章 未雨绸缪	113
试验性工厂和增大规模	116
唯一不变的就是变化本身	117
为变更设计系统	117
为变更计划组织架构	118
前进两步，后退一步	120
前进一步，后退一步	122
第 12 章 干将莫邪	125
目标机器	129
辅助机器和数据服务	131
高级语言和交互式编程	134
第 13 章 整体部分	139
剔除 bug 的设计	142

	构件单元调试	144
	系统集成调试	147
第 14 章	祸起萧墙	153
	里程碑还是沉重的负担	156
	“其他的部分反正会落后”	158
	地毯的下面	159
第 15 章	另外一面	165
	需要什么样的文档	169
	流程图	171
	自文档化的程序	175
第 16 章	没有银弹	181
	摘要	184
	介绍	184
	根本困难	185
	以往解决次要困难的一些突破	190
	银弹的希望	192
	针对概念上根本问题的颇具前途的方法	200
第 17 章	再论“没有银弹”	209
	人狼和其他恐怖传说	212
	存在着银弹——就在这里	212
	含糊的表达将会导致误解	213
	Harel 的分析	216
	Jones 的观点——质量带来生产率	221
	那么，生产率的情形如何	222

面向对象编程——这颗铜质子弹可以吗.....	223
重用的情况怎样	225
学习大量的词汇——对软件重用的一个可预见 但还没有被预言的问题.....	228
子弹的本质——形势没有发生改变	229
第 18 章 《人月神话》的观点：是与非.....	231
第 1 章 焦油坑	234
第 2 章 人月神话.....	235
第 3 章 外科手术队伍.....	236
第 4 章 贵族专制、民主政治和系统设计.....	237
第 5 章 画蛇添足.....	238
第 6 章 贯彻执行.....	239
第 7 章 为什么巴比伦塔会失败	240
第 8 章 胸有成竹.....	242
第 9 章 削足适履.....	243
第 10 章 提纲挈领.....	245
第 11 章 未雨绸缪.....	246
第 12 章 干将莫邪.....	249
第 13 章 整体部分.....	251
第 14 章 祸起萧墙.....	253
第 15 章 另外一面.....	255
第 1 版结束语	256
第 19 章 20 年后的《人月神话》	257
为什么要出版 20 周年纪念版本	260
核心观点——概念完整性和结构师	261

开发第二个系统所引起的后果——盲目的功能 和频率猜测	263
图形界面的成功	265
没有构建舍弃原型——瀑布模型是错误的	269
增量开发模型更佳——渐进地精化	272
关于信息隐藏, Parnas 是正确的, 我是错误的	276
人月到底有多少神话色彩? Boehm 的模型和数据	278
人就是一切(或者说, 几乎是一切)	280
放弃权力的力量	281
最令人惊讶的新事物是什么? 数百万的计算机	283
全新的软件产业——塑料薄膜包装的成品软件	286
买来开发——使用塑料包装的成品软件包作为构件	288
软件工程的状态和未来	290
结束语: 令人向往、激动人心和充满乐趣的 50 年	293
注解与参考文献	295
附录: 人月落地实战体验	315
一、名家谈人月	317
1. 年金	317
2. 《人月神话》与实践	318
3. Frank Chance 评人月	327
4. 软件尚方宝剑(Silver Bullet)何在	330
二、名著评人月	339
三、读者感言	351
1. 读书有感——人月神话	351
2. 我这几天很烦(产品概念完整性)	353

3. 关于我们的思考——“项目开发”及读《人月神话》 有感	355
4. 我的“人月神话”	358
5. 《人月神话》软玉生香	360

CHAPTER



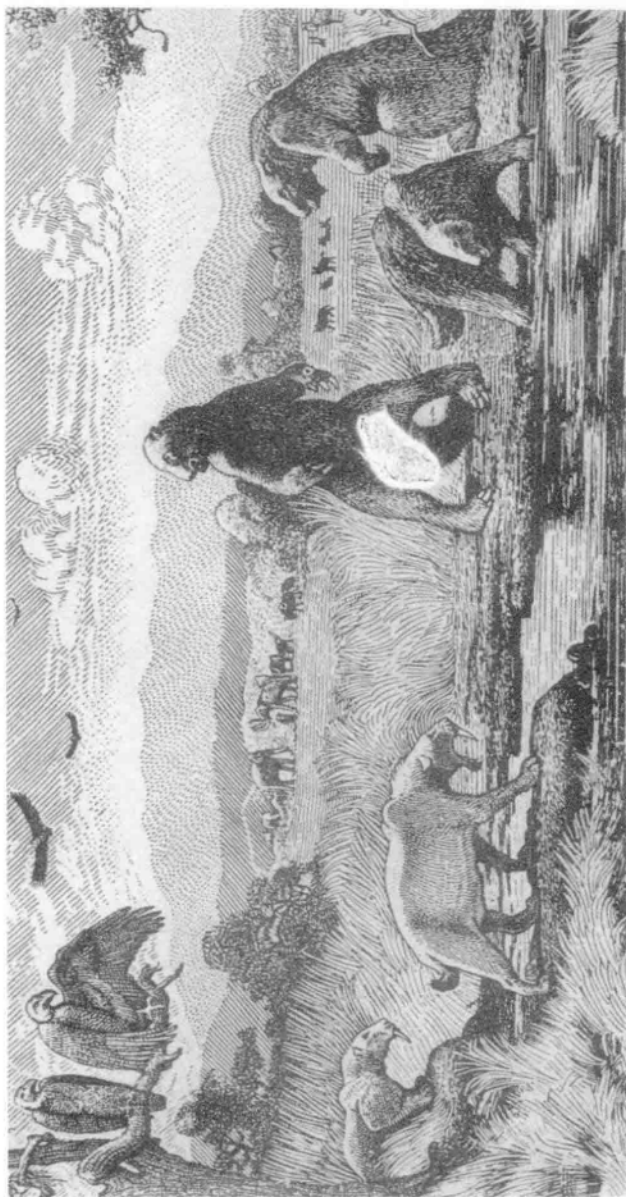
THE Mythical Man-Month

Essays on Software Engineering, Anniversary Edition

第 1 章

焦 油 坑

The Tar Pit



C. R. 奈特的《拉布雷阿的焦油坑壁画》(Mural of La Brea Tar Pits)

注：拉布雷阿焦油坑是著名的旅游景点，位于美国洛杉矶化石博物馆。

资料来源：The George C. Page Museum of La Brea Discoveries, The Natural History Museum of Los Angeles County

前车之覆，后车之鉴。

——荷兰谚语

Een schip op het strand is een baken in zee.

[A ship on the beach is a lighthouse to the sea.]

DUTCH PROVERB

史前史中，没有别的场景比巨兽们在焦油坑中垂死挣扎的场面更令人震撼。上帝见证着恐龙、猛犸象、剑齿虎在焦油中挣扎。它们挣扎得越猛烈，焦油纠缠得就越紧，没有哪种猛兽足够强壮或具有足够的技巧，能够挣脱束缚，它们最后都沉到了坑底。

过去几十年的大型系统开发就犹如这样一个焦油坑，很多大型和强壮的动物在其中剧烈地挣扎。他们中大多数开发出了可运行的系统——不过只有极少数的项目满足了目标、进度和预算的要求。各种团队，大型的或小型的，庞杂的或精干的，一个接一个地淹没在了焦油坑中。表面上看起来好像没有任何一个单独的问题会导致困难，每个问题都能获得解决，但是当它们相互纠缠和累积在一起的时候，团队的行动就会变得越来越慢。对于问题的麻烦程度，每个人似乎都会感到惊讶，并且很难看清问题的本质。不过，如果我们想解决问题，就必须试图先去了解问题。

因此，首先让我们来认识一下系统开发这个职业，以及充满在这个职业中的乐趣和苦恼吧！

■ 编程系统产品

报纸上经常会出现这样的新闻，讲述两个程序员如何在经过改造的简陋车库中，编出超过大型团队工作量的重要程序。接着，每个编程人员准备相信这样的神话，因为他知道自己能以超过产业化团队的 1 000 代码行/年的生产率来开发任何程序。

为什么不是所有的产业化队伍都会被这种专注的二人组合所替代？我们必须看一下产出的是什么。

图 1-1 的左上部分是程序(Program)。它本身是完整的,可以由作者
在所开发的系统平台上运行。它通常是车库中产出的产品,以及作为单
个程序员生产率的评估标准。

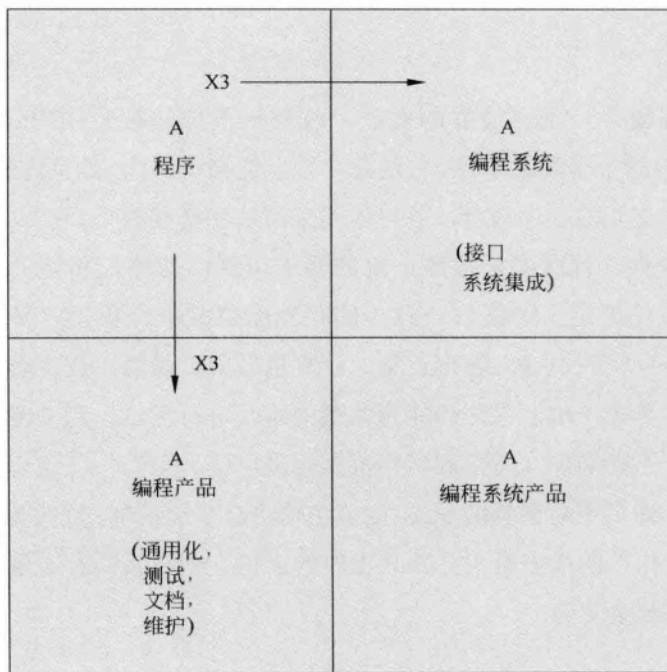


图 1-1 编程系统产品的演进

有两种途径可以使程序转变成更有用但是成本更高的产物,这两种
途径表现为图中的边界。

水平边界以下,程序转变成编程产品(Programming Product)。这是
可以被任何人运行、测试、修复和扩展的程序。它可以在多种操作系统
平台上运行,供多套数据使用。要成为通用的编程产品,程序必须按照
普遍认可的风格来编写,特别是输入的范围和形式必须广泛地适用于所
有可以合理使用的基本算法。接着,对程序进行彻底测试,确保它的稳

定性和可靠性，使其值得信赖。这就意味着必须准备、运行和记录详尽的测试用例库，用来检查输入的边界和范围。此外，要将程序提升为程序产品，还需要有完备的文档，每个人都可以加以使用、修复和扩展。经验数据表明，相同功能的编程产品的成本，至少是已调试的程序的成本的3倍。

回到图中，垂直边界的右边，程序转变成编程系统(Programming System)中的一个构件单元。它是在功能上能相互协作、具有规范的格式、可以进行交互的程序集合，并可以用来组装和搭建整个系统。要成为编程系统构件，程序必须按照一定的要求编制，使输入和输出在语法和语义上与精确定义的接口一致。同时程序还要符合预先定义的资源限制——内存空间、输入输出设备、计算机时间。最后，程序必须同其他系统构件单元一道，以任何能想象到的组合进行测试。由于测试用例会随着组合不断增加，所以测试的范围必须广泛。因为一些意想不到的交互会产生许多不易察觉的 bug，测试工作将会非常耗时，因此相同功能的编程系统构件的成本至少是独立程序的3倍。如果系统有大量的组成单元，成本还会更高。

图 1-1 的右下部分代表编程系统产品(Programming Systems Product)。与以上的所有的简单的程序都不同的是，它的成本高达9倍。然而，只有它才是真正有用的产品，是大多数系统开发的目标。

职业的乐趣

编程为什么有趣？作为回报，它的从业者期望得到什么样的快乐？

首先,这种快乐是一种创建事物的纯粹快乐。如同小孩在玩泥巴时感到快乐一样,成年人喜欢创建事物,特别是自己进行设计。我想这种快乐是上帝创造世界的折射,一种呈现在每片独特的、崭新的树叶和雪花上的喜悦。

其次,这种快乐来自于开发对他人有用的东西。内心深处,我们希望我们的劳动成果能够被他人使用,并能对他们有所帮助。从这一角度而言,这同小孩用粘土为“爸爸的办公室”捏制铅笔盒没有任何本质的区别。

第三,快乐来自于整个过程体现出的一股强大的魅力——将相互啮合的零部件组装在一起,看到它们以精妙的方式运行着,并收到了预期的效果。比起弹球游戏机或自动电唱机所具有的迷人魅力,程序化的计算机毫不逊色。

第四,这种快乐是持续学习的快乐,它来自于这项工作的非重复特性。人们所面临的问题总有这样那样的不同,因而解决问题的人可以从中学习新的事物,有时是实践上的,有时是理论上的,或者兼而有之。

最后,这种快乐还来自于在易于驾驭的介质上工作。程序员,就像诗人一样,几乎仅仅在单纯的思考中工作。程序员凭空地运用自己的想象,来建造自己的“城堡”。很少有创造介质如此灵活,如此易于精炼和重建,如此容易实现概念上的设想(不过我们将会看到,容易驾驭的特性也有它自己的问题)。

然而程序毕竟同诗歌不同,它是实实在在的东西;它可以移动和运行,能独立产生可见的输出;它能打印结果,绘制图形,发出声音,移动支架。神话和传说中的魔术在我们的时代已变成现实。在键盘上键入正确的咒语,屏幕会活动、变幻,显示出前所未有的也不可能存在的事物。

编程的快乐在于它不仅满足了我们内心深处进行创造的渴望，而且还唤醒了每个人内心的情感。

职业的苦恼

然而这个过程并不全都是快乐的。我们只有事先了解一些编程固有的苦恼，这样，当它们真的出现时，才能更加坦然地面对。

首先，苦恼来自追求完美。因为计算机是以这样的方式来变戏法的：如果咒语中的一个字符、一个停顿，没有与正确的形式一致，魔术就不会出现(现实中，很少有人类活动会要求如此完美，所以人类对它本来就不习惯)。实际上，我认为，学习编程最困难的部分，是将做事的方式向追求完美的方向调整^[1]。

其次，苦恼来自由他人来设定目标、供给资源和提供信息。编程人员很少能控制工作环境和工作目标。用管理的术语来说，个人的权威和他所承担的责任是不相配的。不过，似乎在所有的领域中，对要完成的工作，很少能提供与责任相一致的正式权威。而现实情况中，实际(相对于形式)的权威来自于每次任务的完成。

对于系统编程人员而言，对其他人的依赖是一件非常痛苦的事情。他依靠其他人的程序，而这些程序往往设计得并不合理、实现拙劣、发布不完整(没有源代码或测试用例)或者文档记录得很糟。所以，系统编程人员不得不花费时间去研究和修改，而它们在理想情况下本应该是可靠的、完整的。

下一个苦恼——概念性设计是有趣的，但寻找琐碎的 bug 却是一项重复性的活动。伴随着创造性活动的，往往是枯燥沉闷的时间和艰苦的劳动。程序编制工作也不例外。

另外，人们发现调试和查错往往是线性收敛的，或者更糟糕的是，具有二次方的复杂度。结果，测试一拖再拖，寻找最后一个错误比第一个错误将花费更多的时间。

最后一个苦恼，有时也是一种无奈——当投入了大量辛苦的劳动，产品在即将完成或者终于完成的时候，却已显得陈旧过时。可能是同事和竞争对手已在追逐新的、更好的构思；也许替代方案不仅仅是在构思，而且已经在安排了。

现实情况比上面所说的通常要好一些。当产品开发完成时，更优秀的新产品通常还不能投入使用，而仅仅是为大家谈论而已。另外，它同样需要数月的开发时间。事实上，只有实际需要时，才会用到最新的设想，因为所实现的系统已经能满足要求，并体现了回报。

诚然，产品开发所基于的技术在不断地进步。一旦设计被冻结，在概念上就已经开始陈旧了。不过，实际产品需要一步一步按阶段实现。实现落后与否的判断应根据其他已有的系统，而不是未实现的概念。因此，我们所面临的挑战和任务是在实际的进度和有效的资源范围内，寻找解决实际问题的切实可行方案。

这，就是编程，一个许多人痛苦挣扎的焦油坑以及一种乐趣和苦恼共存的创造性活动。对许多人而言，其中的快乐远远大于苦恼。本书的以下章节将试图搭建一些桥梁，为通过这样的焦油坑提供一些指导。

CHAPTER

2

Essays on Software Engineering, Anniversary Edition

THE Mythical Man-Month

第 2 章

人 月 神 话

*The Mythical
Man-Month*

Restaurant Antoine

Fondé En 1840



AVIS AU PUBLIC

Faire de la bonne cuisine demande un certain temps. Si on vous fait attendre, c'est pour mieux vous servir, et vous plaire.

ENTREES (SUITE)

Côtelettes d'agneau grillées 2.50	Entrecôte marchand de vin 4.00
Côtelettes d'agneau aux champignons frais 2.75	Côtelettes d'agneau maison d'or 2.75
Filet de bœuf aux champignons frais 4.75	Côtelettes d'agneau à la parisienne 2.7
Ris de veau à la financière 2.00	Fois de volaille à la brochette 1.50
Filet de bœuf nature 3.75	Tournedos nature 2.75
Tournedos Médicis 3.25	Filet de bœuf à la hawaïenne 4.00
Pigeonneaux sauce paradis 3.50	Tournedos à la hawaïenne 3.25
Tournedos sauce béarnaise 3.25	Tournedos marchand de vin 3.25
Entrecôte minute 2.75	Pigeonneaux grillés 3.00
Filet de bœuf béarnaise 4.00	Entrecôte nature 3.75
Tripes à la mode de Caen (commander d'avance) 2.00	Châteaubriand (30 minutes) 7.0

LÉGUMES

Epinards sauce crème .60	Chou fleur au gratin .60
Broccoli sauce hollandaise .80	Asperges fraîches au beurre .90
Pommes de terre au gratin .60	Carottes à la crème .60
Haricots verts au beurre .60	Pommes de terre soufflées .6
Petits pois à la française .75	

SALADES

Salade Antoine .60	Fonds d'artichauts Bayard .9
Salade Mirabeau .75	Salade de laitue aux œufs .60
Salade laitue au roquefort .80	Tomate frottée à la Jules César .60
Salade de laitue aux tomates .60	Salade de cœur de palmier 1.00
Salade de légumes .60	Salade aux pointes d'asperges .60
Salade d'anchois 1.00	Avocat à la vinaigrette .60

DESSERTS

Gâteau moka .50	Cerises jubilé 1.25
Meringue glacée .60	Crêpes à la gelée .80
Crêpes Suzette 1.25	Crêpes nature .70
Glace sauce chocolat .60	Omelette au rhum 1.10
Fruits de saison à l'eau-de-vie .75	Glace à la vanille .50
Omelette soufflée à la Jules César (2) 2.00	Fraises au kirach .9
Omelette Alaska Antoine (2) 2.50	Pêche Melba .6

FROMAGES

Roquefort .50	Liederkranz .50	Gruyère .50
Camembert .50	Fromage à la crème Philadelphia .50	

CAFÉ ET THÉ

Café .20	Café au lait .20	Thé .20
Café brûlot diabolique 1.00	Thé glacé .20	Demi-tasse .1

EAUX MINÉRALES—BIÈRE—CIGARES—CIGARETTES

White Rock	Cliquot Club	Bière locale	Canada Dry	Cigares
Vichy				Cigarettes



Roy L. Alcatorre, Propriétaire

718-717 Rue St. Louis

Nouvelle Orléans, Louisiane

新奥尔良市安托万餐厅的菜单

注：安托万餐厅成立于1840年。本章的序言即来自这张菜单。

美食的烹调需要时间；片刻等待，更多美味，更多享受。

——新奥尔良市安托万餐厅的菜单

Good cooking takes time. If you are made to wait, it is to serve you better, and to please you.

MENU OF RESTAURANT ANTOINE, NEW ORLEANS

在众多软件项目中，缺乏合理的进度安排是造成项目滞后的最主要原因，它比其他所有因素加起来的影响还要大。导致这种灾难如此普遍的原因是什么呢？

首先，我们对估算技术缺乏有效的研究，更加严肃地说，它反映了一种悄无声息但并不真实的假设——一切都将运作良好。

第二，我们采用的估算技术隐含地假设人和月可以互换，错误地将进度与工作量相互混淆。

第三，由于对自己的估算缺乏信心，软件经理通常不会有耐心持续地估算这项工作。

第四，对进度缺少跟踪和监督。在其他工程领域中，经过验证的跟踪技术和常规监督程序，在软件工程中常常被认为是大胆的革新。

第五，当意识到进度的偏移时，下意识(以及传统)的反应是增加人力。这就像使用汽油灭火一样，只会使事情更糟。越来越大的火势需要更多的汽油，从而进入了一场注定会导致灾难的循环。

进度监督是另一篇论文的主题，而本文中我们将对这一问题的其他方面进行更详细的讨论。

■ 乐观主义

所有的编程人员都是乐观主义者。可能是这种现代魔术特别吸引那些相信美满结局和幻想中的圣母的人；也可能是成百上千琐碎的挫折赶走了大多数人，只剩下了那些习惯上只关注结果的人；还可能仅仅因为计算机还很年轻，程序员更加年轻，而年轻人总是些乐观主义者——无

论是什么样的程序，结果是毋庸置疑的：“这次它肯定会运行”，或者“我刚刚找出最后一个错误”。

所以系统编程的进度安排背后的第一个错误的假设是：一切都将运作良好，每一项任务仅花费它所“应该”花费的时间。

对这种弥漫在编程人员中的乐观主义，理应受到慎重的分析。Dorothy Sayers 在她的《创造者的思想》(*The Mind of the Maker*)一书中，将创造性活动分为三个阶段：构思、实现和交流。书籍、计算机或者程序的存在，首先是作为一个构思或模型出现在作者的脑海中，它与时间和空间无关。接着，借助钢笔、墨水和纸，或者电线、硅片和铁氧体，在现实的时间和空间中实现它们。然后，当某人阅读书籍、使用计算机和运行程序的时候，他与作者的思想相互沟通，创造过程从而得以结束。

以上 Sayers 的阐述不仅仅可以描绘人类的创造性活动，而且类似于“基督三位一体的教义”，能指导我们的日常工作。对于创造者，只有在实现的过程中，才能发现我们构思的不完整性和不一致性。因此，对于理论家而言，书写、试验以及“工作实现”是非常基本和必要的。

在许多创造性活动中，往往很难掌握活动实施的介质，例如木头切割、油漆、电气接线等。这些介质的物理约束限制了思路的表达，它们同样对实现造成了许多预料之外的困难。

由于物理介质和思路中隐含的不完善性，实现起来需要花费大量的时间和汗水。对遇到的大部分实现上的困难，我们总是倾向于去责怪那些物理介质，因为它们不顺应“我们”设定的思路。其实，这只不过是我们的骄傲使判断带上了主观主义色彩。

然而，计算机编程基于十分容易掌握的介质，编程人员通过非常纯粹的思维活动——概念以及灵活的表现形式来开发程序。正是由于介质

的易于驾驭，我们期待在实现过程中不会碰到困难，因此造成了乐观主义的弥漫。而我们的构思是有缺陷的，因此总会发现 bug。也就是说，我们的乐观主义并不应该是理所应当的。

在单个的任务中，“一切都将运转正常”的假设在进度上具有可实现性。因为所遇到的延迟是一个概率分布曲线，“不会延迟”具有有限的概率，所以现实情况可能会像计划安排的那样顺利。然而大型的编程工作，或多或少包含了很多任务，某些任务间还具有前后的次序，从而一切正常的概率变得非常小，甚至接近于零。

人月

第二个谬误的思考方式是在估计和进度安排中使用的工作量单位：人月。成本的确随开发产品的人数和时间的不同，有着很大的变化，进度却不是如此。人员和时间的关系如图 2-1 所示。因此我认为用人月作为衡量一项工作的规模是一个危险和带有欺骗性的神话。它暗示着人员数量和时间是可以相互替换的。

人数和时间的互换仅仅适用于以下情况：某个任务可以分解给参与人员，并且他们之间不需要相互的交流(见图 2-1)。这在割小麦或收获棉花的工作中是可行的；而在系统编程中近乎不可能。

当任务由于次序上的限制不能分解时，人手的添加对进度没有帮助(见图 2-2)。无论哪位母亲，孕育一个生命都需要 10 个月。由于除错的次序特性，许多软件开发工作都具有这种特征。

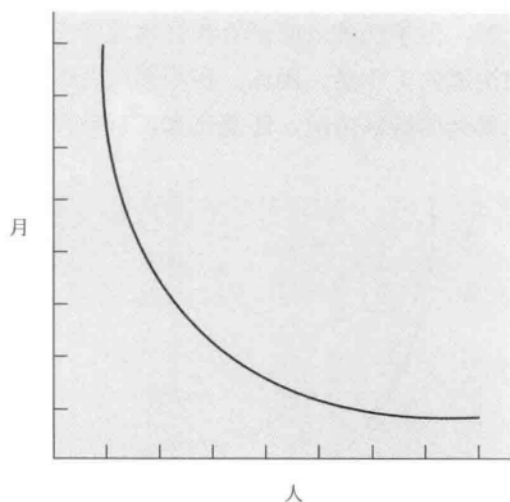


图 2-1 人员和时间之间的关系——完全可以分解的任务

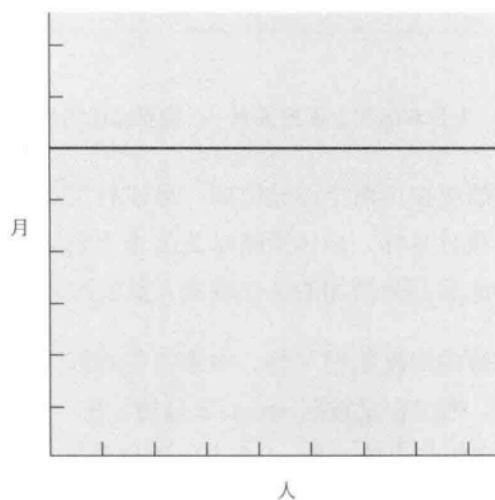


图 2-2 人员和时间之间的关系——无法分解的任务

对于可以分解，但子任务之间需要相互沟通和交流的任务，必须在计划工作中考虑沟通的工作量。因此，在相同人月的前提下，采用增加人手来减少时间得到的最好情况，还是比未调整前差一些(见图 2-3)。

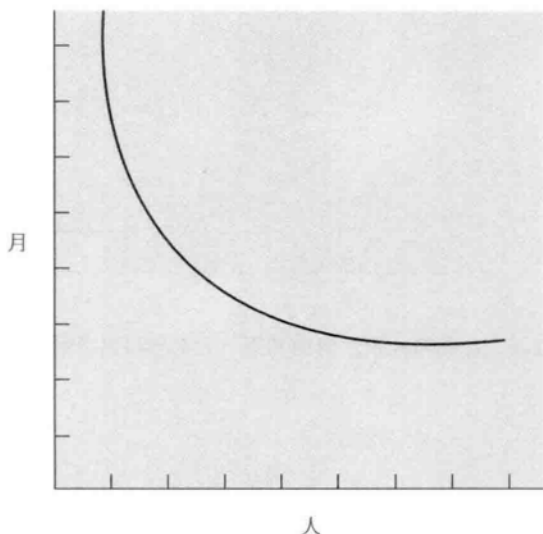


图 2-3 人员和时间之间的关系——需要沟通的可分解任务

沟通所增加的负担由两个部分组成：培训和相互的交流。每个成员需要进行技术、项目目标、总体策略以及工作计划的培训。这种培训是不能分解的，因此这部分增加的工作量随人员的数量呈线性变化^[1]。

相互之间交流的情况更糟一些。如果任务的每个部分必须分别与其他部分单独协作，则工作量按照 $n(n-1)/2$ 递增。在一对一交流的情况下，3 个人的工作量是 2 个人的 3 倍，4 个人的工作量则是 2 个人的 6 倍。而对于需要在三、四个人之间召开会议、进行协商、一同解决的问题，情况会更加恶劣。所增加的用于沟通的工作量可能会完全抵消对原有任务分解所产生的作用，此时我们会被带到如图 2-4 所示的境地。

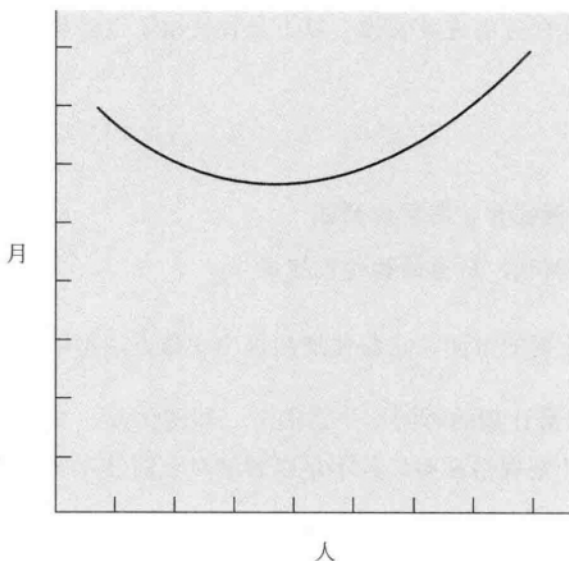


图 2-4 人员和时间之间的关系——关系错综复杂的任务

因为软件开发本质上是一项系统工作——错综复杂关系下的一种实践，沟通、交流的工作量非常大，它很快会消耗任务分解所节省下来的个人时间。从而，添加更多的人手，实际上是延长了而不是缩短了时间进度。

系统测试

在进度安排中，由于顺序限制所造成的影响，没有哪个部分比单元测试和系统测试所受到的牵涉更彻底。而且，需要的时间依赖于所遇到的错误、缺陷的数量及其难以捕捉的程度。理论上，缺陷的数量应该为零。但是，由于我们的乐观主义，通常实际出现的缺陷数量比预料的要多得多。因此，系统测试进度的安排常常是编程中最不合理的部分。

对于软件任务的进度安排，以下是我使用了很多年的经验法则：

1/3 计划

1/6 编码

1/4 构件测试和早期系统测试

1/4 系统测试，所有的构件已完成

在许多重要的方面，它与传统的进度安排方法不同：

(1) 分配给计划的时间比平常的多。即便如此，其只是勉强产生详细和稳定的计划规格说明，并不足以容纳对全新技术的研究和探索；

(2) 对所完成代码的调试和测试投入近一半的时间，这比平常的安排多很多；

(3) 容易估计的部分，如编码，仅仅分配了 1/6 的时间。

通过对传统项目进度安排的研究，我发现很少有项目允许为测试分配一半的时间，但大多数项目的测试实际上是花费了进度中一半的时间。它们中的许多项目，在系统测试之前还能保持进度。或者说，除了系统测试，进度基本能够保证^[2]。

特别需要指出的是，不为系统测试安排足够的时间简直就是一场灾难。因为延迟发生在项目快完成的时候。直到接近项目的发布日期，才有人发现进度上的问题。因此，坏消息没有任何预兆，很晚才出现在客户和项目经理面前。

另外，此时此刻的延迟具有不寻常的、严重的财务和心理上的反应。在此之前，项目已经配置了充足的人员，每天的人力成本也已经达到了最大的限度。更为严重的是，在用软件支持其他的商业活动(计算机硬件运

送、新设备操作等)的情况下,若在即将发布时出现延误,所付出的二次商业代价是非常高昂的。实际上,上述的二次成本远远高于其他开销。因此,在早期进度策划时,允许充分的系统测试时间是非常重要的。

空泛的估算

观察一下编程人员,你可能会发现,同厨师一样,某项任务的计划进度,可能受限于顾客要求的紧迫程度,但紧迫程度无法控制实际的完成情况。就像约好在两分钟内完成一个煎蛋,看上去可能进行得非常好,但当它无法在两分钟内完成时,顾客只能选择等待或者生吃煎蛋。软件顾客的情况类似。

厨师还有其他的选择:他可以把火开大,不过结果常常是无法“挽救”的煎蛋——一面已经焦了,而另一面还是生的。

现在,我并不认为软件经理内在的勇气和坚定不如厨师,或者不如其他工程经理。但为了满足顾客期望的日期而造成的不合理进度安排,在软件领域中却比其他任何工程领域要普遍得多。而且,非阶段化方法的采用,少得可怜的数据支持,加上完全借助软件经理的直觉,这样的方式很难生产出有力的、看似可靠的和规避风险的估计。

显然我们需要两种解决方案。开发并推行生产率图表、缺陷率图表、估算规则等,而整个组织最终会从这些数据的共享上获益。

或者,在基于可靠基础的估算出现之前,项目经理需要挺直腰杆,坚持他们的估计,确信自己的经验和直觉总比从期望派生出的结果要强得多。

重复产生的进度灾难

当一个软件项目落后于进度时，通常的做法是什么呢？自然是加派人手。如图 2-1、图 2-2、图 2-3、图 2-4 所示，这可能有所帮助，却可能无法解决问题。

我们来考虑一个例子^[3]。设想一个估计需要 12 个人月的任务，分派给 3 个成员在 4 个月的时间内完成，并在每个月的末尾安排了可测量的里程碑 A、B、C、D(见图 2-5)。

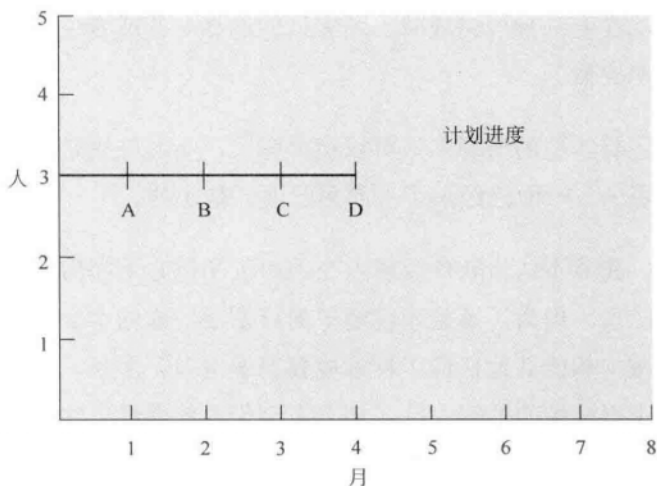


图 2-5 4 个里程碑

现在假定两个月之后，第一个里程碑没有达到(见图 2-6)。项目经理面对的选择方案有哪些呢？

(1) 假设任务必须按时完成。假设仅仅是任务的第一个部分估计不得当，如图 2-6 所示，则剩余了 9 个人月的工作量，时间还有两个月，即需要 4.5 个开发人员，所以需要在原来 3 个人的基础上增加 2 个人。

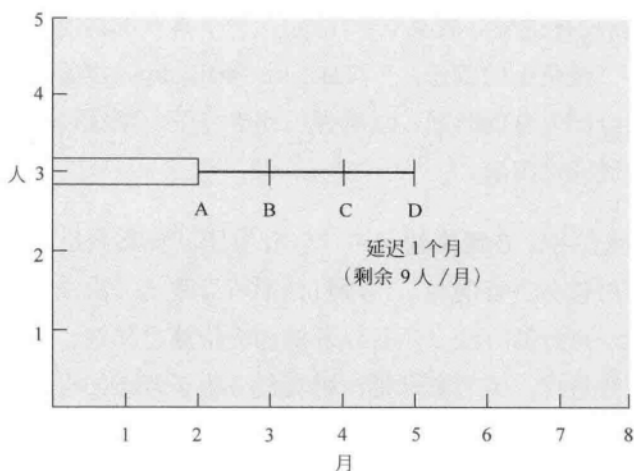


图 2-6 2 个月后，第一个里程碑没有达到

(2) 假设任务必须按时完成。假设整个任务的估计偏少，如图 2-7 所示，那么还有 18 个人月的工作量，这样就需要 2 个月的时间和 9 个人，比原来计划的 3 人多了 6 人。

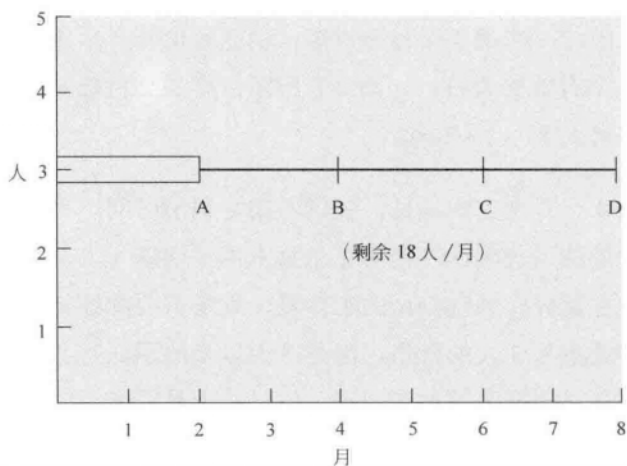


图 2-7 估计偏少的情况

(3) 重新安排进度。我喜欢 P. Fagg, 一个具有丰富经验的硬件工程师的忠告: “避免小的偏差。” (Take no small slips.) 也就是说, 在新的进度安排中分配充分的时间, 以确保工作能仔细、彻底地完成, 从而无需重新确定时间进度表。

(4) 削减任务。在现实情况中, 当开发团队观察到进度的偏差时, 总是倾向于对任务进行削减。当项目延期所导致的二次成本非常高时, 这常常是唯一可行的方法。项目经理的相应措施是仔细、认真地调整项目, 重新安排进度; 或者默默地注视着任务由于轻率的设计和不完整的测试而被剪除。

在前两种情况下, 坚持把不经调整的任务在 4 个月内完成将是灾难性的。考虑到重复生成的工作量, 以第一种为例(见图 2-8)——不论在多么短的时间内, 聘请到两名多么能干的新员工, 他们都需要接受一位有经验的职员的培训。如果培训需要一个月的时间, 那么 3 个人月将会投入到原有进度安排以外的工作中。另外, 原先划分为 3 个部分的工作, 会重新分解成 5 个部分; 某些已经完成的工作必定会丢失, 系统测试必然被延长。因此, 在第 3 个月的月末, 仍然残留着 7 个人月的工作, 但此时只有 5 个有效的人月。如图 2-8 所示, 产品交付还是会延期, 如同没有增加任何人手(见图 2-6)。

期望在 4 个月内完成项目, 仅仅考虑培训的时间, 不考虑任务的重新划分和额外的系统测试, 在第 2 个月末需要增添 4 人, 而不是 2 人。如果考虑任务划分和系统测试的工作量, 则还需要继续增加人手。到那时所拥有的就不是 3 人的队伍, 而是 7 人以上的团队; 并且小组的组织 and 任务的划分在类型上都不尽相同, 这已经不是程度上的差异问题。

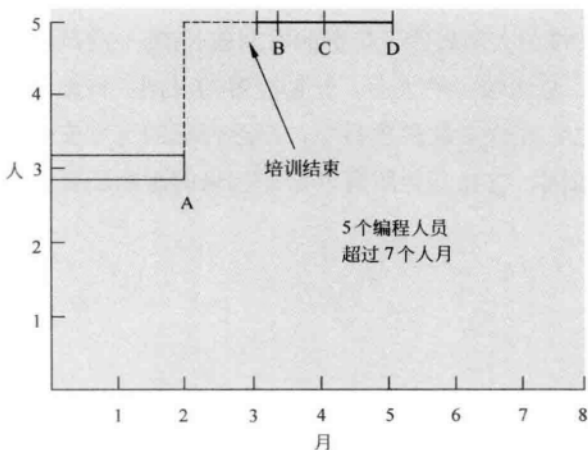


图 2-8 重复的工作量

注意在第3个月的结尾时，情况看上去非常糟。尽管已经做出了所有的管理方面的努力，3月1日的里程碑仍未达到。此时，对项目经理而言，仍然存在着很强的诱惑——添加更多人力，结果往往会是上述情况的循环重复。这简直就是一种疯狂、愚蠢的做法。

前面的讨论仅仅是第一个里程碑估计不当的情况。如果在3月1日，项目经理做出了比较保守的假设，即整个估计过于乐观了，如图2-7所示，6个人手需要添加到原先的任务中。这里，我们把培训、任务的重新分配、系统测试工作量的计算作为练习留给读者。但是毫无疑问，重复“灾难”所开发出的产品，比没有增加人手，而是重新安排开发进度所产生的产品更差。

冒昧地简化一下 Brooks 法则：

向进度落后的项目中增加人手，只会使进度更加落后。(Adding manpower to a late software project makes it later.)

这就是除去了神话色彩的人月。项目的时间依赖于顺序上的限制，人员的最大数量依赖于独立子任务的数量。从这两个数值可以推算出进

度表，该表安排的人员较少，花费的时间较长(唯一的风险是产品可能会过时)。相反，分派较多的人手，计划较短的时间，将无法得到可行的进度安排。总之，在众多软件项目中，缺乏合理的进度安排是造成项目滞后的最主要原因，它比其他所有因素加起来的影响还要大。

CHAPTER

3

THE Mythical Man-Month

Essays on Software Engineering, Anniversary Edition

第 3 章

外科手术队伍

The Surgical Team



外科手术现场

资料来源：UPI[United Press International] Photo/The Bettman Archive

这些研究表明，效率高和效率低的实施者之间个体差异非常大，经常能够达到数量级的水平。

——*Sackman, Erikson 和 Grant*^[1]

These studies revealed large individual differences between high and low performers, often by an order of magnitude.

SACKMAN, ERIKSON, AND GRANT¹

提供计算机、IT 类 pdf 电子版代找服务，如果你找不到自己想要的书的 pdf 电子版，我们可以帮您找到，如有需要，请联系 QQ 23846268.

声明：本人只提供代找服务，每本 100%索引书签和目录，因寻找 pdf 电子书有一定难度，仅收取代找费用。

如因 PDF 产生的版权纠纷，与本人无关，我们仅仅只是帮助你寻找到你要的 pdf 而已。

因 PDF 电子书都有版权，请不要随意传播，如果您有经济购买能力，请尽量购买正版。

鉴于很多朋友需要复制或搜索 PDF 中的文字，本人提供转换服务，绝大多数 PDF 都可转换成可复制、可搜索内容的 PDF、清晰度等质量与原 PDF 文件保持一致。

注意，不是转换成 word 等，是把不可复制、搜索内容的 PDF 转换成可搜索、复制内容的 PDF。