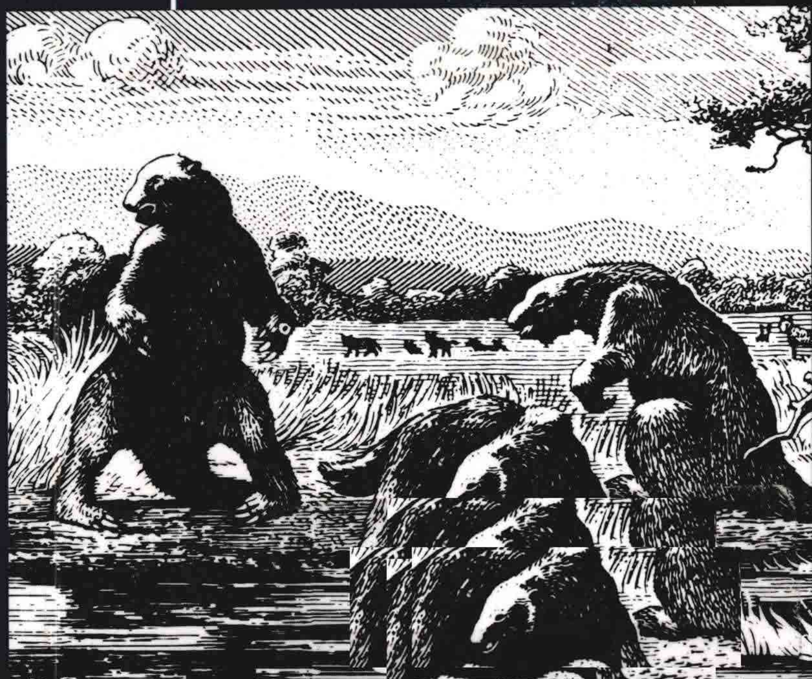
TURING 图灵程序设计丛书

Addison Wesley



The Mythical Man-Month

Essays on Software Engineering

人月神话（英文版）

[美] Frederick P. Brooks, Jr. 著

- 图灵奖得主总结软件项目实战得失
- 软件工程领域35年畅销不衰的经典
- 软件从业人员不可不读的宝书

人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

人月神话（英文版）

[美] Frederick P. Brooks, Jr. 著

人民邮电出版社
北　京

图书在版编目（CIP）数据

人月神话 = The Mythical Man-Month: Essays on Software Engineering ：英文 /（美）布鲁克斯(Brooks, F.P.) 著. -- 北京 ：人民邮电出版社，2010.8
（图灵程序设计丛书）
ISBN 978-7-115-23268-7

Ⅰ. ①人… Ⅱ. ①布… Ⅲ. ①软件开发－英文 Ⅳ. ①TP311.52

中国版本图书馆CIP数据核字(2010)第122117号

内 容 提 要

本书内容源于作者Brooks在IBM公司任System/360计算机系列以及其庞大的软件系统OS/360项目经理时的实践经验。在本书中，Brooks为人们管理复杂项目提供了最具洞察力的见解，既有很多发人深省的观点，又有大量软件工程的实践，为每个复杂项目的管理者给出了自己的真知灼见。

大型编程项目深受由于人力划分产生的管理问题的困扰，保持产品本身的概念完整性是一个至关重要的需求。本书探索了达成一致性的困难和解决的方法，并探讨了软件工程管理的其他方面。本书适合任何软件开发行业的从业人员阅读，对软件开发人员、软件项目经理、系统分析师更是必读之作。

图灵程序设计丛书
人月神话（英文版）

◆ 著　　[美] Frederick P. Brooks, Jr.
责任编辑　杨海玲

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街14号
邮编　100061　　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京隆昌伟业印刷有限公司印刷

◆ 开本：880×1230　1/32
印张：10.5
字数：268千字　　2010年8月第1版
印数：1－2 500册　　2010年8月北京第1次印刷

著作权合同登记号　图字：01-2006-5373号

ISBN 978-7-115-23268-7

定价：29.00元

读者服务热线：(010)51095186　印装质量热线：(010)67129223
反盗版热线：(010)67171154

版权声明

Original edition, entitled *The Mythical Man-Month: Essays on Software Engineering*, 9780201835953 by Frederick P. Brooks, Jr., published by Pearson Education, Inc., publishing as Addison-Wesley, Copyright ©1995 by Addison Wesley Longman, Inc.

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.

China edition published by PEARSON EDUCATION ASIA LTD. and POSTS & TELECOM PRESS Copyright © 2010.

This edition is manufactured in the People's Republic of China, and is authorized for sale only in the People's Republic of China excluding Hong Kong, Macao and Taiwan.

本书英文版由 Pearson Education Asia Ltd. 授权人民邮电出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

仅限于中华人民共和国境内（香港、澳门特别行政区和台湾地区除外）销售发行。

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

版权所有，侵权必究。

20 周年纪念版前言

20 年过去了,《人月神话》依然受到读者喜欢,真让我惊喜不已,这本书已总计印行 25 万多册了。常常有人问我，1975 年阐述的那些观点和建议中，哪些我仍坚持如故，哪些已有所改变以及如何改变。尽管我已经时不时地在各种演讲场合提到过这些问题，但我一直期望能专门撰文来进行说明。

Peter Gordon 自 1980 年以来一直耐心地和我一起工作，对我颇有助益，现在他已是 Addison-Wesley 的出版合伙人了。他提议出版《人月神话》的纪念版。我们决定不对初版进行改动，而是只做一些细微的校正，基本上是将原内容重印，同时增补几章更现代的思想。

第 16 章重刊了 1986 年我在 IFIPS（国际信息处理学会联合会）上发表的论文《没有银弹：软件工程的必然和偶然》（*No Silver Bullet: Essence and Accidents of Software Engineering*），这篇论文源自我主持一项国防科学委员会军事软件研究时的经验。论文的合作者兼我们的执行秘书 Robert L. Patrick 发挥了不可估量的作用，他让我重新亲历了真实的大型软件项目。此文曾于 1987 年重新刊发于 IEEE《计算机》杂志上，从而广为传播。

《没有银弹》引起了轩然大波，它预言在 10 年内不会出现任何编程技术能够给软件生产率带来数量级上的提高。再有 1 年就满 10 年了,我的预言看来是安全了。从大家发表的文章来看,相比《人月神话》,《没有银弹》激发的讨论越来越热烈。因而，第 17 章评议了发表过的一些评论意见，并更新了我 1986 年提出的这个观点。

试读结束：需要全本请在线购买： www.ertongbook.com

在着手回顾和更新《人月神话》的过程中，我发现其中断言的观点很少被软件工程研究和实践所评判、证实或驳斥，这令我颇为震动。因此我觉得现在有必要把这些观点归类整理出来，而剔除掉相关的分析数据。我把这些论点列为第18章，希望这些简单的陈述可以抛砖引玉，引发大家以论据或事实来证实、驳斥、更新或细化这些观点。

第19章是对旧日文章的更新。提请读者注意的是，这些新观点不像初版那样来自一线实践经验。我后来一直在大学中工作，而不是工业界，所接触的也是小规模的项目，而非大项目。自1986年以来，我只是教授软件工程课程，而完全不作这方面的研究。我的研究转向了虚拟环境及其应用。

准备这一版期间，我咨询了实际从事软件工程工作的朋友们。我非常感谢他们慷慨地与我分享观点，深思熟虑地对书稿提出意见，令我重获教益。他们是：Barry Boehm、Ken Brooks、Dick Case、James Coggins、Tom DeMarco、Jim McCarthy、David Parnas、Earl Wheeler和Edward Yourdon。此外，Fay Ward出色地完成了新章节的排版制作。

对于本书的第16章，我要感谢Gordon Bell、Bruce Buchanan、Rick Hayes-Roth，他们是我在国防科学委员会军事软件任务组中的同事，还要特别感谢David Parnas，是他们的真知灼见启发了我写这篇文章；我还要感谢Rebekah Bierly完成了这篇论文的排版制作。用“必然和偶然”的方法来分析软件问题则是受到了Nancy Greenwood Brooks的启发，她在一篇关于铃木小提琴教学的论文中使用了这样的分析法。

在1975年版的前言中，我按出版社当时的规矩不能鸣谢其职员起到的重要作用。在此特别表扬两个人的贡献：执行主编Norman Stanton和美术总监Herbert Boes。Boes制作了典雅的风格，曾经有个评论人特地赞扬道：“宽大的页边距、字体的使用和排版布局颇具创意。”更重要的是，是他建议每一章要有一幅图片作为

开篇。（当时我只有焦油坑和兰斯大教堂的图片。）找齐这些图片多花了我一年的时间，不过我对他的建议无限感激。

Soli Deo gloria—— 愿神独得荣耀。

Frederick P. Brooks, Jr.
北卡罗来纳州查布尔希尔市
1995年3月

第 1 版前言

管理大型计算机编程项目在很多方面与管理大型集体事务类似——比大多数程序员能想到的方面还要多，但它又在许多其他方面与后者不同——同样比大多数职业经理人能想到的方面还要多。

这个领域的知识日益增长，已经有了几个与此相关的会议，AFIPS（美国信息处理学会联合会）会议增设了一些讨论组，还出版或发表了一些书籍和论文。但是真正系统化的教科书还未出现。因而推出这个小册子似乎正当其时，它基本上反映了我的个人观点。

尽管我最初是在计算机编程方面成长起来的，但在自动控制程序和高级语言编译器发展壮大的数年中（1956~1963），我主要参与的工作却是硬件体系结构。故而在 1964 年，当我出任操作系统 OS/360 的经理时，我发现编程世界经过几年后已经发生了巨大的变化。

管理 OS/360 开发是一个很有意义的经历，也是非常令我受挫的经历。这个团队，包括我的继任者 F. M. Trapnell 在内，有许多值得骄傲的地方。系统在设计和运行方面皆有许多出色之处，它成功地实现了广泛使用的目标。某些思想，比如最引人注目的设备无关的输入输出和外部库管理，现在已经成为被广泛采纳的技术创新。如今这一系统已然相当可靠，具有不错的效率，并且十分通用。

然而，上述工作不能说是完全的成功。每个 OS/360 的用户很快就会发现它有提高的余地。设计和执行错误散布在与语言编译器分离的控制程序里。多数此类错误可追溯到 1964 至 1965 年的设计

阶段，故而我难逃其咎。而且这个产品的发布延迟了，它比原计划使用了更多的内存，总开销也数倍于原来的预算。开始时它还不能运转得很好，直到几个发布版本之后才得到转机。

在接手 OS/360 时就已说好，于是 1965 年我离开 IBM 来到查布尔希尔市的北卡罗来纳大学，我开始分析 OS/360 的经验，看看应该吸取哪些管理和技术方面的教训。我尤其希望好好分析一下 System/360 硬件开发和 OS/360 软件开发中所遭遇的完全不同的管理体验。Tom Watson 曾苦苦探究为何编程难以管理，本书算是对此问题的一个迟来的答案。

在探索过程中，我与 1964 至 1965 年的项目助理经理 R. P. Case 和 1965 至 1968 年的经理 F. M. Trapnell 有过数次长谈，使我获益匪浅。我将结论与其他巨型编程项目经理交换了意见，包括麻省理工学院的 F. J. Corbato、贝尔电话实验室的 John Harr 和 V. Vyssotsky、国际计算机有限公司的 Charles Portman、苏联科学院西伯利亚分部计算实验室的 A.P. Ershov 以及 IBM 的 A.M. Pietrasanta。

我本人的结论收录于本书的文章中，以飨各位职业程序员和职业经理，尤其是管理程序员的职业经理们。

各章尽管独立成文，但都体现了一个中心观点，尤其是第 2 章至第 7 章。简言之，我相信大型编程项目面临的管理问题与小型项目不同，主要是由于人力划分产生的问题。我相信保持产品本身的概念完整性是一个至关重要的要求。这几章既分析了达成一致性的困难，又探讨了解决的方法。后续的各章则探讨了软件工程管理的其他方面。

这一领域的文献并不多，却很分散。因而我努力给出参考文献，帮助阐释特定的观点，也为感兴趣的读者指出其他有用的著作。许多朋友已经读过了手稿，有些人提出了大量颇有帮助的意见，有些内容不便于放到正文中，我将在注解（Notes）中提到。

由于这是一本文集而不是教科书，所有的参考文献和注释都放到全书末尾，我主张读者第一遍阅读时略过它们。

我衷心地感谢 Sara Elizabeth Moore 小姐、David Wagner 先生和 Rebecca Burris 夫人在本书写作期间给予我的帮助，并感谢 Joseph C. Sloane 教授对插图的建议。

Frederick P. Brooks, Jr.

北卡罗来纳州查布尔希尔市

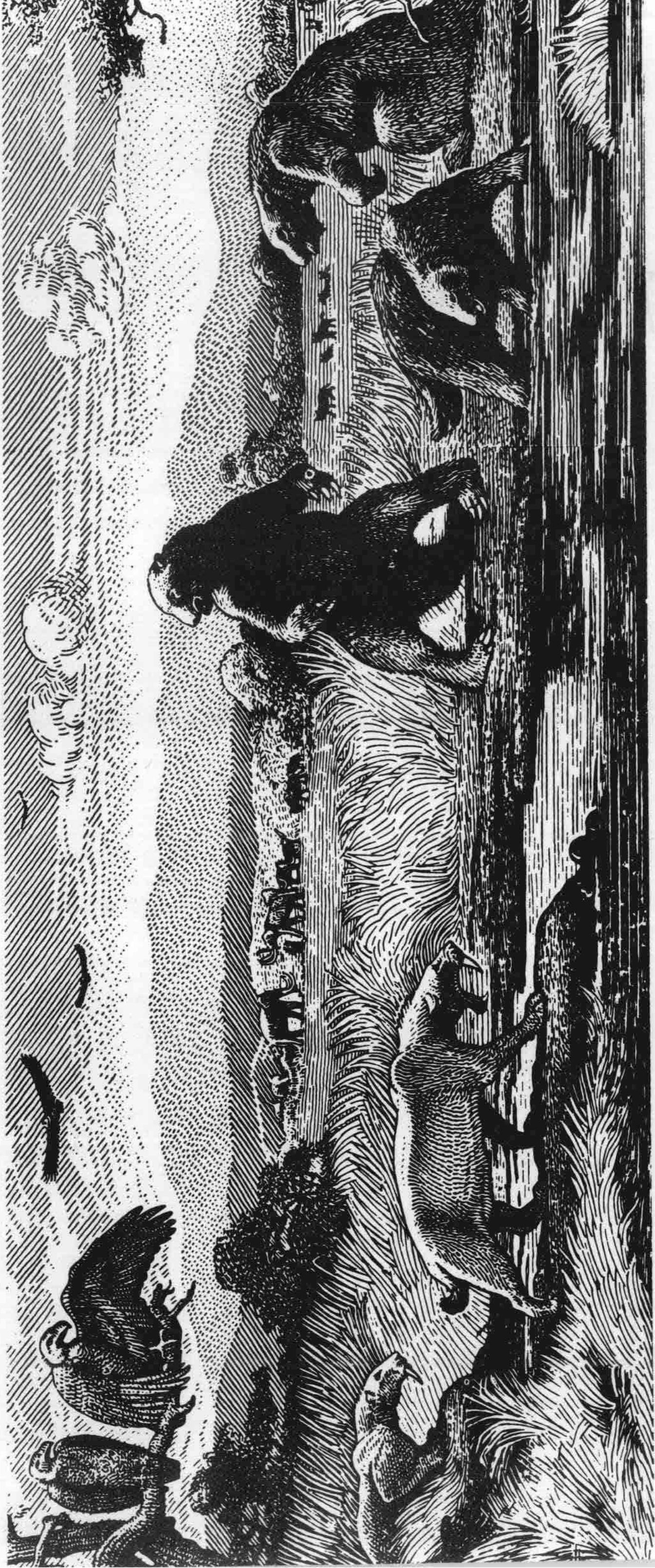
1974 年 10 月

Contents

Chapter 1	The Tar Pit	3
Chapter 2	The Mythical Man-Month	13
Chapter 3	The Surgical Team	29
Chapter 4	Aristocracy, Democracy, and System Design	41
Chapter 5	The Second-System Effect	53
Chapter 6	Passing the Word	61
Chapter 7	Why Did the Tower of Babel Fail?	73
Chapter 8	Calling the Shot	87
Chapter 9	Ten Pounds in a Five-Pound Sack	97
Chapter 10	The Documentary Hypothesis	107
Chapter 11	Plan to Throw One Away	115
Chapter 12	Sharp Tools	127
Chapter 13	The Whole and the Parts	141
Chapter 14	Hatching a Catastrophe	153
Chapter 15	The Other Face	163
Chapter 16	No Silver Bullet—Essence and Accident	177
Chapter 17	"No Silver Bullet" Refired	205
Chapter 18	Propositions of *The Mythical Man-Month:* True or False?	227
Chapter 19	*The Mythical Man-Month* after 20 Years	251
	Epilogue	291
	Notes and References	293
	Index	309

试读结束：需要全本请在线购买： www.ertongbook.com

1
The Tar Pit



1
The Tar Pit

Een schip op het strand is een baken in zee.
[A ship on the beach is a lighthouse to the sea.]

DUTCH PROVERB

C. R. Knight, Mural of La Brea Tar Pits
The George C. Page Museum of La Brea Discoveries,
The Natural History Museum of Los Angeles County

No scene from prehistory is quite so vivid as that of the mortal struggles of great beasts in the tar pits. In the mind's eye one sees dinosaurs, mammoths, and sabertoothed tigers struggling against the grip of the tar. The fiercer the struggle, the more entangling the tar, and no beast is so strong or so skillful but that he ultimately sinks.

Large-system programming has over the past decade been such a tar pit, and many great and powerful beasts have thrashed violently in it. Most have emerged with running systems—few have met goals, schedules, and budgets. Large and small, massive or wiry, team after team has become entangled in the tar. No one thing seems to cause the difficulty—any particular paw can be pulled away. But the accumulation of simultaneous and interacting factors brings slower and slower motion. Everyone seems to have been surprised by the stickiness of the problem, and it is hard to discern the nature of it. But we must try to understand it if we are to solve it.

Therefore let us begin by identifying the craft of system programming and the joys and woes inherent in it.

The Programming Systems Product

One occasionally reads newspaper accounts of how two programmers in a remodeled garage have built an important program that surpasses the best efforts of large teams. And every programmer is prepared to believe such tales, for he knows that he could build *any* program much faster than the 1000 statements/year reported for industrial teams.

Why then have not all industrial programming teams been replaced by dedicated garage duos? One must look at *what* is being produced.

In the upper left of Fig. 1.1 is a *program.* It is complete in itself, ready to be run by the author on the system on which it was developed. *That* is the thing commonly produced in garages, and

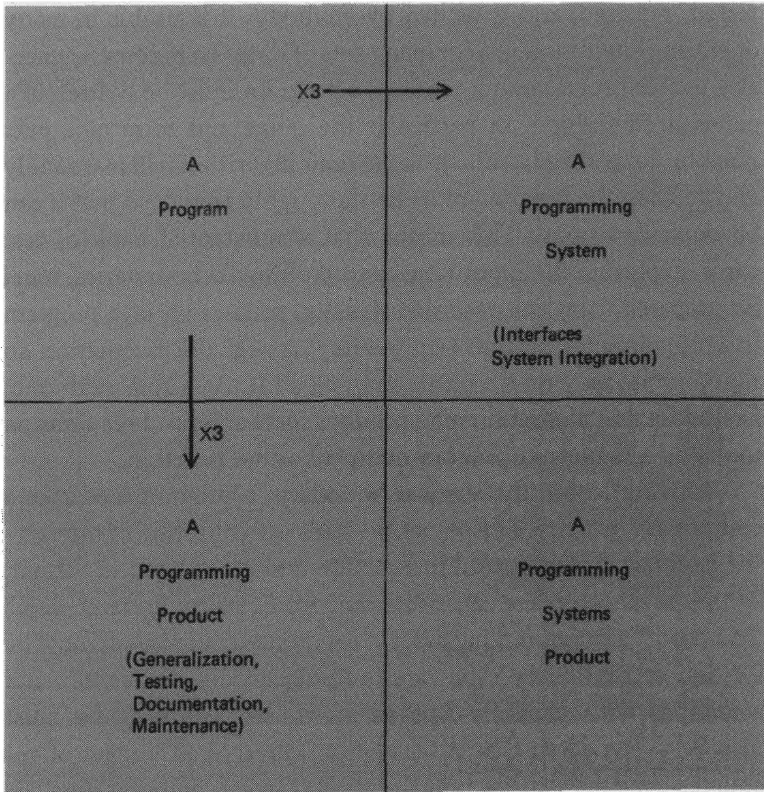


Fig. 1.1 Evolution of the programming systems product

that is the object the individual programmer uses in estimating productivity.

There are two ways a program can be converted into a more useful, but more costly, object. These two ways are represented by the boundaries in the diagram.

Moving down across the horizontal boundary, a program becomes a *programming product.* This is a program that can be run,

tested, repaired, and extended by anybody. It is usable in many operating environments, for many sets of data. To become a generally usable programming product, a program must be written in a generalized fashion. In particular the range and form of inputs must be generalized as much as the basic algorithm will reasonably allow. Then the program must be thoroughly tested, so that it can be depended upon. This means that a substantial bank of test cases, exploring the input range and probing its boundaries, must be prepared, run, and recorded. Finally, promotion of a program to a programming product requires its thorough documentation, so that anyone may use it, fix it, and extend it. As a rule of thumb, I estimate that a programming product costs at least three times as much as a debugged program with the same function.

Moving across the vertical boundary, a program becomes a component in a *programming system.* This is a collection of interacting programs, coordinated in function and disciplined in format, so that the assemblage constitutes an entire facility for large tasks. To become a programming system component, a program must be written so that every input and output conforms in syntax and semantics with precisely defined interfaces. The program must also be designed so that it uses only a prescribed budget of resources—memory space, input-output devices, computer time. Finally, the program must be tested with other system components, in all expected combinations. This testing must be extensive, for the number of cases grows combinatorially. It is time-consuming, for subtle bugs arise from unexpected interactions of debugged components. A programming system component costs at least three times as much as a stand-alone program of the same function. The cost may be greater if the system has many components.

In the lower right-hand corner of Fig. 1.1 stands the *programming systems product.* This differs from the simple program in all of the above ways. It costs nine times as much. But it is the truly useful object, the intended product of most system programming efforts.

试读结束：需要全本请在线购买： www.ertongbook.com