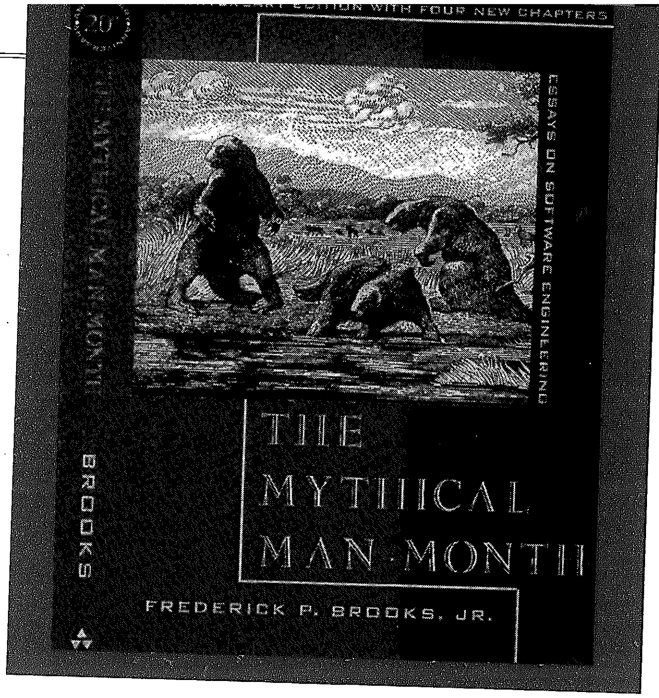
on
I S S R E
5

neral Chair
:arama Kanoun
LAAS-CNRS
, Avenue du Colonel Roche
1077 Toulouse Cedex, France
:anoun@laas.fr
Tel: + (33) 61 33 6235
Fax: + (33) 61 33 6411
gram Co-Chairs
Mitsuru Ohba
Hiroshima City University, Japan
hba@edu.ipc.hiroshima-cu.ac.jp
Fax: + (81) 82 830 1657
Mladen A. Vouk
North Carolina State Univ., USA
ouk@adm.csc.ncsu.edu
Fax :+(1) 919 515 7896 or 6497
blication Chair
Mohamed Kaâniche
LAAS-CNRS
aaniche@laas.fr
Tel: +(33) 61 33 6405
Fax: +(33) 61 33 6411
cal Arrangements
Alain Dreuil, SITEF, France
Marie-Thérèse Ippolito, LAAS-CNRS
oëlle Penavayre, LAAS-CNRS
torials Co-Chairs
Fevzi Belli
University of Paderborn, Germany
elli@uni-paderborn.de
Fax: +(49) 525 160 3246
George J. Knafl
De Paul University, USA
nafl@cs.depaul.edu
Fax: +(1) 312 362 6116
ols Fair Co-Chairs
Yves Crouzet
LAAS-CNRS
rouzet@laas.fr
Fax: +(33) 61 33 6411
Richard M. Karcich /IEEE Rep.
StorageTek, USA
Rick_Karcich@stortek.com
Fax: +(1) 303 673 8641
ustry Liaison Chair
Patricia Mangan
Aerospace Corpo., USA
patricia_mangan@macmail.aero.org
Fax: +(1) 310 336 3442
gram Committee
Ackerman (USA)
B. Bastani (USA)
Chillarege (USA)
Ehrenberger (D)
Faisandier (F)
C. Gaudel (F)
Goševa-Popstojanova (Mac.)
Hamlet (USA)
Horgan (USA)
K. Iyer (USA)
Jones (USA)
Khoshgoftaar (USA)
M. Lapassat (F)
C. Laprie (F)
Littlewood (UK)
Lyu (USA)
K. Malaiya (USA)
Matsumoto (JP)
von Mayrhauser (USA)
C. Munson (USA)
D. Musa (USA)
Nara (J)
Nikora (USA)
Pasquini (IT)
Sakamoto (J)
Schneidewind (USA)
Seviora (CDN)
Stark (USA)
Tohma (J)
Vallée (F)
Voas (USA)
Wohlin (SW)
Xie (SG)

and LAAS-CNRS. Organize
eliability Engineering of the
on Software Engineering
Support from AT&T, BNR,



The Mythical Man-Month

AFTER 20 YEARS

Frederick P. Brooks, Jr.

Excerpted from Chapter 19, "The Mythical Man-Month After 20 Years," in The Mythical Man-Month: Anniversary Edition, *copyright © 1995 by Addison-Wesley Publishing Company, Inc. Reprinted with permission.*

Artwork courtesy The George C. Page Museum of La Brea Discoveries, The Natural History Museum of Los Angeles County.

WHY IS THERE A TWENTIETH ANNIVERSARY EDITION?

The plane droned through the night toward LaGuardia. Clouds and darkness veiled all interesting sights. The document I was studying was pedestrian. I was not, however, bored. The stranger sitting next to me was reading *The Mythical Man-Month*, and I was waiting to see if by word or sign he would react. Finally as we taxied toward the gate, I could wait no longer:

"How is that book? Do you recommend it?"

"Hmph! Nothing in it I didn't know already."

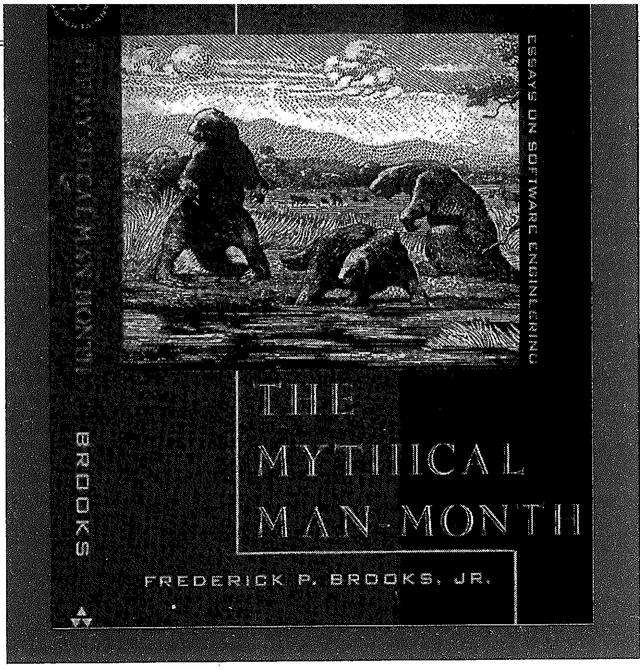
I decided not to introduce myself.

Why has *The Mythical Man-Month* persisted? Why is it still seen to be relevant to software practice today? Why does it have a readership outside the software engineering community, generating reviews, citations, and correspondence from lawyers, doctors, psychologists, sociologists, as well as from software people? How can a book written 20 years ago about a software-building experience 30 years ago still be relevant, much less useful?

One explanation sometimes heard is that the software development discipline has not advanced normally or properly. This view is often supported by contrasting computer software development productivity with computer hardware manufacturing productivity, which has multiplied at least a thousandfold over the two decades. As Chapter 16 explains, the anomaly is not that software has been so slow in its progress but rather that computer technology has exploded in a fashion unmatched in human history. By and large this comes from the gradual transition of computer manufacturing from an assembly industry to a process industry, from labor-intensive to capital-intensive manufacturing. Hardware and software development, in contrast to manufacturing, remain inherently labor-intensive.

A second explanation often advanced is that *The Mythical Man-Month* is only incidentally about software but primarily about how people in teams make things. There is surely some truth in this; in the preface to the 1975 edition I said that managing a software project is more like other management than most programmers initially believe. I still believe that to be true. Human history is a drama in which the stories stay the same, the scripts of those stories change slowly with evolving cultures, and the stage settings change all the time. So it is that we see our twentieth-century selves mirrored in Shakespeare, Homer, and the Bible. So to the extent that *The MM-M* is about people and teams, obsolescence should be slow.

Whatever the reason, readers continue to buy the book, and they continue to send me much-appreciated comments. Nowadays I am often asked, "What do you think was wrong when written? What is now obsolete? What is really new in the software engineering world?" These quite distinct questions are all fair, and I shall address them as best I can. Not in that order, however, but in clusters of topics.



PARNAS WAS RIGHT AND I WAS WRONG ABOUT INFORMATION HIDING

In Chapter 7 I contrast two approaches to the question of how much each team member should be allowed or encouraged to know about the designs and code of other team members. In the Operating System/360 project, we decided that *all* programmers should see *all* material, i.e., each programmer having a copy of the project workbook, which came to number over 10,000 pages. Harlan Mills has argued persuasively that "programming should be a public process," that exposing all the work to everybody's gaze helps quality control both by peer pressure to do things well and by peers actually spotting flaws and bugs.

This view contrasts sharply with David Parnas's teaching that modules of code should be encapsulated with well-defined interfaces, and that the interior of such a module should be the private property of its programmer, not discernible from outside. Programmers are most effective if shielded from, not exposed to, the innards of modules not their own.

I dismissed Parnas's concept as a "recipe for disaster" in Chapter 7. Parnas was right, and I was wrong. I am now convinced that information hiding, today often embodied in object-oriented programming, is the only way of raising the level of software design.

One can indeed get disasters with either technique. Mills' technique ensures that programmers can know the detailed semantics of the interfaces they work to by knowing what is on the other side. Hiding those semantics leads to system bugs. On the other hand, Parnas's technique is robust under change and is more appropriate in a design-for-change philosophy. Chapter 16 argues the following:

♦ Most past progress in software productivity has come from eliminating noninherent difficulties such as awkward machine languages and slow batch turnaround.

♦ There are not a lot more of these easy pickings.

♦ Radical progress is going to have to come from attacking the essential difficulties of fashioning complex conceptual constructs.

The most obvious way to do this recognizes that programs are made up of conceptual chunks much larger than the individual high-level language statement — subroutines, or modules, or classes. If we can limit design and building so that we only do the putting together and parameterization of such chunks from prebuilt collections, we have radically raised the conceptual level, and eliminated the vast amounts of work and the copious opportunities for error that dwell at the individual statement level.

Parnas's information-hiding definition of modules is the first published step in that crucially important research program, and it is an intellectual ancestor of object-oriented programming. He defined a module as a software entity with its own data model and its own set of operations. Its data can only be accessed via one of its proper operations. The second step was a contribution of several thinkers: the upgrading of the Parnas module into an *abstract data type*, from which many objects could be derived. The abstract data type provides a uniform way of thinking about and specifying module interfaces, and an access discipline that is easy to enforce.

The third step, object-oriented programming, introduces the powerful concept of *inheritance*, whereby classes (data types) take as defaults specified attributes from their ancestors in the class hierarchy. Most of what we hope to gain from object-oriented programming derives in fact from the first step, module encapsulation, plus the idea of prebuilt libraries of modules or classes *that are designed and tested for reuse.* Many people have chosen to ignore the fact that such modules are not just programs, but instead are program products in the sense discussed in Chapter 1. Some people are vainly hoping for significant module reuse without paying the initial cost of building product-quality modules — generalized, robust, tested, and documented. Object-oriented programming and reuse are discussed in Chapters 16 and 17.

PEOPLE ARE EVERYTHING (WELL, ALMOST EVERYTHING)

Some readers have found it curious that *The MM-M* devotes most of the essays to the managerial aspects of software engineering, rather than the many technical issues. This bias was due in part to the nature of my role on the IBM Operating System/360 (now MVS/370). More fundamentally, it sprang from a conviction that the quality of the people on a project, and their organization and management, are much more important factors in success than are the tools they use or the technical approaches they take.

Subsequent researches have supported that conviction. Boehm's COCOMO model finds that the quality of the team is by far the largest factor in its success, indeed four times more potent than the next largest factor. Most academic research on software engineering has concentrated on tool. I admire and covet sharp tools. Nevertheless, it is encouraging to see ongoing research efforts on the care, growing, and feeding of people, and on the dynamics of software management.

Peopleware. A major contribution during recent years has been DeMarco and Lister's 1987 book, *Peopleware: Productive Projects and Teams*. Its underlying thesis is that "The major problems of our work are not so much *technological* as *sociological* in nature." It abounds with gems such as, "The manager's function is not to make people work, it is to make it possible for people to work." It deals with such mundane topics as space, furniture, team meals together. DeMarco and Lister provide real data from their Coding War Games that show stunning correlation between performances of programmers from the same organization, and between workplace characteristics and both productivity and defect levels.

The top performers' space is quieter, more private, better protected against interruption, and there is more of it.... Does it really matter to you ... whether quiet, space, and privacy help your current people to do better work or [alternatively] help you to attract and keep better people?

I heartily recommend the book to all my readers.

Moving projects. DeMarco and Lister give considerable attention to team *fusion*, an intangible but vital property. I think it is management's overlooking fusion that accounts for the readiness I have observed in multilocation companies to move a project from one laboratory to another.

My experience and observation are limited to perhaps a half-dozen moves. I have never seen a successful one. One can move *missions* successfully. But in every case of attempts to move projects, the new team in fact started over, in spite of having good documentation, some well-advanced designs, and some of the people from the sending team. I think it is the breaking of fusion of the old team that aborts the embryonic product, and brings about restart.

THE POWER OF GIVING UP POWER

If one believes, as I have argued at many places in this book, that creativity comes from individuals and not from structures or processes, then a central question facing the software manager is how to design structure and process so as to enhance, rather than inhibit, creativity and initiative. Fortunately, this problem is not peculiar to software organizations, and great thinkers have worked on it. E.F. Schumacher, in his classic, *Small is Beautiful: Economics as if People Mattered*, proposes a theory of organizing enterprises to maximize the creativity and joy of the workers. For his first principle he chooses the "Principle of Subsidiary Function" from the Encyclical *Quadragesimo Anno* of Pope Leo XIII:

It is an injustice and at the same time a grave evil and disturbance of right order to assign to a greater and higher association what lesser and subordinate organizations can do. For every social activity ought of its very nature to furnish help to the members of the body social and never destroy and absorb them.... Those in command should be sure that the more perfectly a graduated order is preserved among the various associations, in observing the principle of subsidiary function, the stronger will be the social authority and effectiveness and the happier and more prosperous the condition of the State.

Schumacher goes on to interpret:

The Principle of Subsidiary Function teaches us that the centre will gain in authority and effectiveness if the freedom

Interact
With Us

Visit the
IEEE Software
Home Page
for a look at
where we've been,
where we're headed,
and how you can be a part
of our future.

Our Web site includes

- Abstracts from current and past issues
- A calendar of upcoming theme issues and calls for papers
- Guidelines for writing and submitting articles
- Staff and editorial board contact information
- An author and subject index for 1994-1995
- Information on copyrights, reprints, and subscriptions

and more . . .

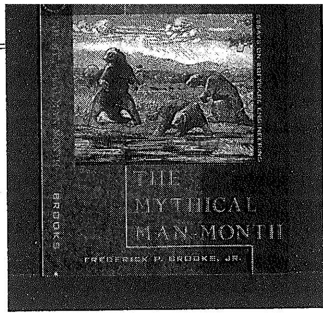
Visit the IEEE Software Home Page directly at:

http://www.computer.org/pubs/software/software.htm

Or access our page via the IEEE Computer Society Home Page at:

http://www.computer.org/

IEEE Software



and responsibility of the lower formations are carefully preserved, with the result that the organization as a whole will be "happier and more prosperous."

How can such a structure be achieved? ... The large organization will consist of many semi-autonomous units, which we may call quasi-firms. *Each of them will have a large amount of freedom, to give the greatest possible chance to creativity and* entrepreneurship.... *Each quasi-firm must have both a profit and loss account, and a balance sheet.*

Among the most exciting developments in software engineering are the early stages of putting such organizational ideas into practice. First, the microcomputer revolution created a new software industry of hundreds of start-ups. These firms, all of them starting small, and marked by enthusiasm, freedom, and creativity. The industry is changing now, as many small companies continue to be acquired by larger ones. It remains to be seen if the larger acquirers will understand the importance of preserving the creativity of smallness.

More remarkably, high management in some large firms have undertaken to delegate power down to individual software project teams, making them approach Schumacher's quasi-firms in structure and responsibility. They are surprised and delighted at the results.

Jim McCarthy of Microsoft described to me his experience at emancipating his teams:

Each feature team (30-40 people) owns its feature set, its schedule, and even its process of how to define, build, ship. The team is made up for four or five specialties, including building, testing, and writing. The team settles squabbles; the bosses don't. I can't emphasize enough the importance of empowerment, of the team being accountable to itself for its success.

Earl Wheeler, retired head of IBM's software business, told me his experience in undertaking the downward delegation of power long centralized in IBM's division managements:

The key thrust [of recent years] was delegating power down. It was like magic! Improved quality, productivity, morale. We have small teams, with no central control. The teams own the process, but they have to have one. They have many different processes. They own the schedule, but they feel the pressure of the market. This pressure causes them to reach for tools on their own.

Conversations with individual team members, of course, show both an appreciation of the power and freedom that is delegated, and a somewhat more conservative estimate of how much control really is relinquished. Nevertheless, the delegation achieved is clearly a step in the right direction. It yields exactly the benefits Leo XIII predicted: the center gains in real authority by delegating power, and the organization as a whole is happier and more prosperous. ◆