Labor Dynamics of the IT Economy
What IT Planners Need to Know About the Nature of Programming

Eric Roberts
Professor of Computer Science
Stanford University

U.S. State Department
IT Strategy Conference
San Francisco
November 18, 2004

General Thesis

In terms of the dynamics of work in the field, computing is different from most engineering domains. Assumptions that hold true in other, more traditional disciplines often turn out to be wrong when applied to computing, particularly when a project requires a significant software-development effort. Understanding these differences is essential to the task of formulating effective technology policy.

General Thesis

In terms of the dynamics of work in the field, computing is different from most engineering domains. Assumptions that hold true in other, more traditional disciplines often turn out to be wrong when applied to computing, particularly when a project requires a significant software-development effort. Understanding these differences is essential to the task of formulating effective technology policy.

My intent in this session is to provide an overview of those characteristics of computing that policymakers need to comprehend in order to make appropriate decisions.

Bridging Cultural Gaps

Once upon a time, a group of farmers asked a mathematician to help them increase the yield of their dairy herd.

Bridging Cultural Gaps

Once upon a time, a group of farmers asked a mathematician to help them increase the yield of their dairy herd.

The mathematician went away to study the problem and came back after a time with a report.

The report began with the words:

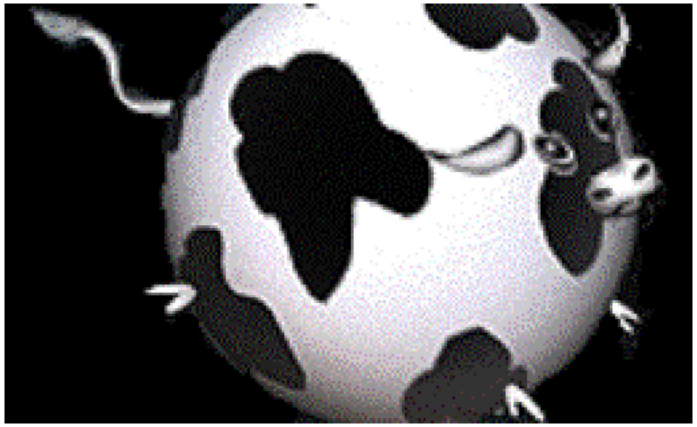
Bridging Cultural Gaps

Once upon a time, a group of farmers asked a mathematician to help them increase the yield of their dairy herd.

The mathematician went away to study the problem and came back after a time with a report.

The report began with the words:

Assuming a spherical cow. . .



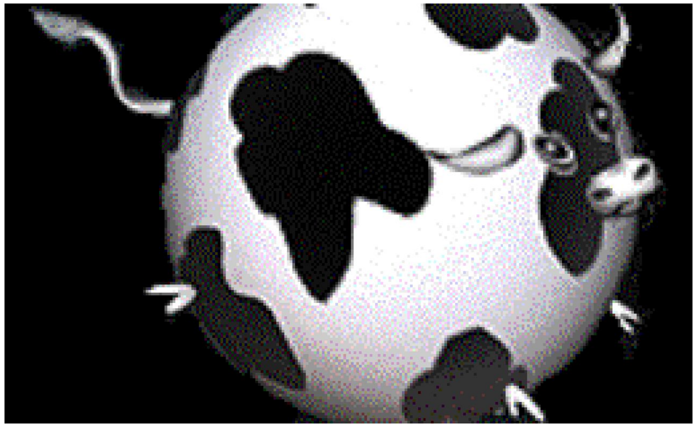
Bridging Cultural Gaps

Once upon a time, a group of farmers asked a mathematician to help them increase the yield of their dairy herd.

The mathematician went away to study the problem and came back after a time with a report.

The report began with the words:

Assuming a spherical cow. . .



Of course, if a mathematician were to ask farmers for advice, the results might be even more laughable, such as $\pi = 3$.

An Illustrative Example

Because of the preconceptions they have from other disciplines, university administrators often have considerably difficulty understanding the dynamics of faculty recruitment in computer science. If I quote the 1999-2000 ACM finding that

There is one candidate for every three faculty positions.

many listeners hear this statistic backwards. Deans and presidents who are used to having hundreds of applicants for any open job show signs of disbelief and ask:

Are there really only three candidates per position?

The idea that there might be <u>fewer</u> applicants than positions simply does not register.

Critical Observations about Software

1. Software development is an extraordinarily difficult task, exceeding in complexity most other engineering work. That difficulty, moreover, is intrinsic to the discipline and is not likely to change in the foreseeable future.

2. Software development requires people with an unusual combination of skills. Those people are in short supply, but their economic value is huge. Experienced programmers differ in productivity by several orders of magnitude.

3. Economic, social, and political factors are more important than technological progress in determining how computing evolves.

Critical Observations about Software

1. Software development is an extraordinarily difficult task, exceeding in complexity most other engineering work. That difficulty, moreover, is intrinsic to the discipline and is not likely to change in the foreseeable future.

2. Software development requires people with an unusual combination of skills. Those people are in short supply, but their economic value is huge. Experienced programmers differ in productivity by several orders of magnitude.

3. Economic, social, and political factors are more important than technological progress in determining how computing evolves.

The Difficulty of Software

People familiar with both software engineering and older engineering disciplines observe that the state of the art in software is significantly behind that in other areas of engineering. When most engineering products have been completed, tested, and sold, it is reasonable to expect that the product design is correct and that it will work reliably. With software products, it is usual to find that the software has major "bugs" and does not work reliably for some users.

— David L. Parnas, *Software Aspects c*
Strategic Defense Systems, 1985

NMD Software Is Particularly Hard

Specifying, generating, testing, and maintaining the software for a battle management system will be a task that far exceeds in complexity and difficulty any that has yet been accomplished in the production of civil or military software systems.

— *The Fletcher Report on Battl*
*Management,*February 1984

What Makes Software Different?

What Makes Software Different?

- Computers are used to solve hard problems.

What Makes Software Different?

- Computers are used to solve hard problems.
- Software has high “system complexity” and is therefore difficult to distribute among members of a large team.

Brooks's Law

"Adding manpower to a late software project makes it later."
— Fred Brooks, *The Mythical Man Month*

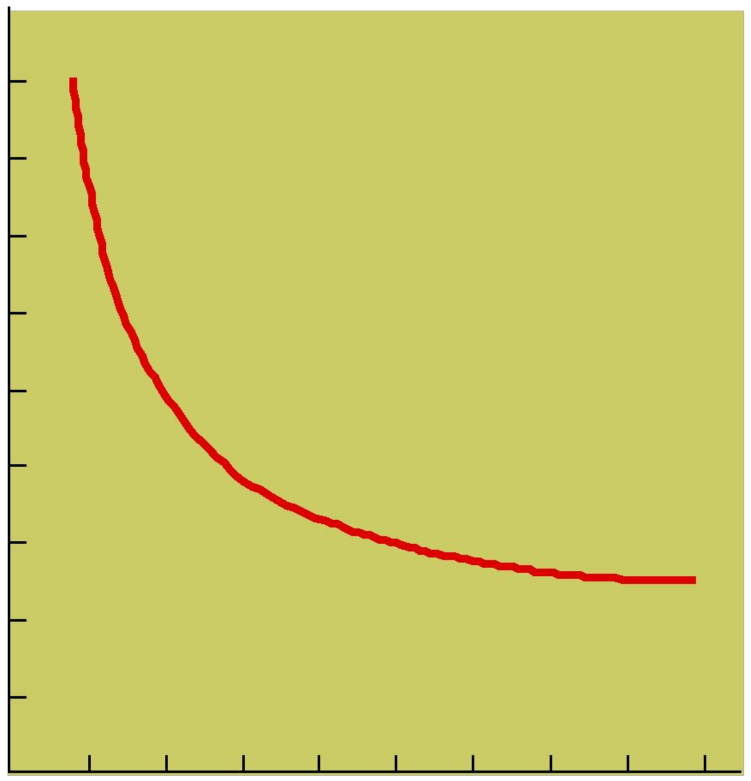


Fig. 2.3 Time versus number of workers—partitionable task requiring communication

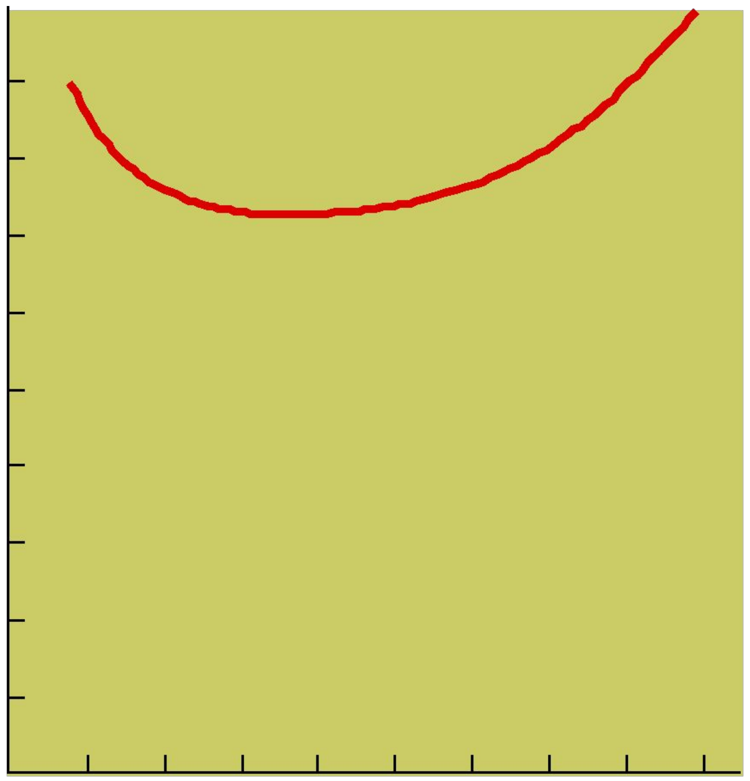


Fig. 2.4 Time versus number of workers—task with complex interrelationships

What Makes Software Different?

- Computers are used to solve hard problems.
- Software has high "system complexity" and is therefore difficult to distribute among members of a large team.
- Bugs are everpresent and inevitable.

The Inevitability of Bugs

Although programming techniques have improved immensely since the early days, the process of finding and correcting errors in programming known graphically—if inelegantly—as debugging still remains a most difficult, confused and unsatisfactory operation. . . . Although we are happy to pay lip-service to the adage that to err is human, most of us like to make a small private reservation about our own performance on special occasions when we really try. It is somewhat deflating to be shown publicly and incontrovertibly by a machine that even when we do try, we in fact make just as many mistakes as other people. If your pride cannot recover from this blow, you will never make a programmer.

— Christopher Strachey, *Scientific American*, 1966

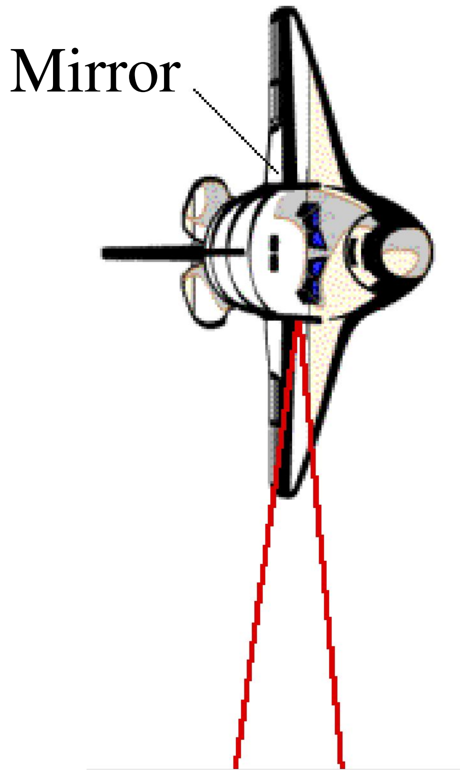
Even in National Missile Defense

Simply because of its inevitable large size, the software capable of performing the battle management task for strategic defense will contain errors. All systems of useful complexity contain software errors.

— Eastport report on Computing in Support of Battle Management, December 1985

The Space Shuttle Laser Test

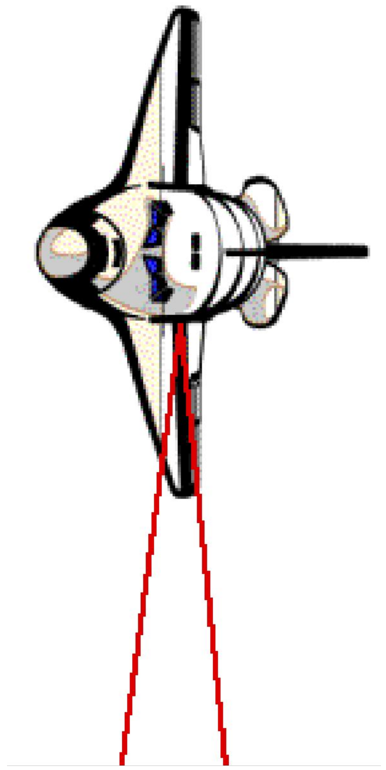
On June 19, 1985, one of the first Star Wars tests failed because the altitude of a ground-based laser was entered in feet instead of nautical miles. [New York Times, July 20, 1985]



The Space Shuttle Laser Test

On June 19, 1985, one of the first Star Wars tests failed because the altitude of a ground-based laser was entered in feet instead of nautical miles. [New York Times, July 20, 1985]

They got it right the second time around.



What Makes Software Different?

- Computers are used to solve hard problems.
- Software has high “system complexity” and is therefore difficult to distribute among members of a large team.
- Bugs are everpresent and inevitable.
- Software systems are discrete rather than continuous: it is impossible to “overengineer” such systems to ensure safety.

What Makes Software Different?

- Computers are used to solve hard problems.
- Software has high "system complexity" and is therefore difficult to distribute among members of a large team.
- Bugs are everpresent and inevitable.
- Software systems are discrete rather than continuous: it is impossible to "overengineer" such systems to ensure safety.
- Software systems are inherently chaotic: small changes in initial conditions generate massive changes in the results.

What Makes Software Different?

- Computers are used to solve hard problems.
- Software has high “system complexity” and is therefore difficult to distribute among members of a large team.
- Bugs are everpresent and inevitable.
- Software systems are discrete rather than continuous: it is impossible to “overengineer” such systems to ensure safety.
- Software systems are inherently chaotic: small changes in initial conditions generate massive changes in the results.
- The discipline of software engineering has not had centuries in which to mature.

Computing Is a New Discipline

We find it a bit troublesome to be discussing whether radical advancements in software technology would enhance the quality of a new defense system, when we are aware that many of the DoD's biggest software development contractors are presently literally decades behind the state of the art—an art that is only a few decades old.

— Eastport report on Computing in Support of Battle Management, December 1985

What Makes Software Different?

- Computers are used to solve hard problems.
- Software has high "system complexity" and is therefore difficult to distribute among members of a large team.
- Bugs are everpresent and inevitable.
- Software systems are discrete rather than continuous: it is impossible to "overengineer" such systems to ensure safety.
- Software systems are inherently chaotic: small changes in initial conditions generate massive changes in the results.
- The discipline of software engineering has not had centuries in which to mature.

Critical Observations about Software

1. Software development is an extraordinarily difficult task, exceeding in complexity most other engineering work. That difficulty, moreover, is intrinsic to the discipline and is not likely to change in the foreseeable future.

2. Software development requires people with an unusual combination of skills. Those people are in short supply, but their economic value is huge. Experienced programmers differ in productivity by several orders of magnitude.

3. Economic, social, and political factors are more important than technological progress in determining how computing evolves.

Variations in Programmer Productivity

- In 1968, a study by Sackman, Erikson, and Grant revealed that programmers with the same level of experience exhibit variations of more than 20 to 1 in the time required to solve particular programming problems.

Variations in Programmer Productivity

- In 1968, a study by Sackman, Erikson, and Grant revealed that programmers with the same level of experience exhibit variations of more than 20 to 1 in the time required to solve particular programming problems.

- More recent studies [Curtis 1981, DeMarco and Lister 1985, Brian 1997] confirm this high variability.

Variations in Programmer Productivity

- In 1968, a study by Sackman, Erikson, and Grant revealed that programmers with the same level of experience exhibit variations of more than 20 to 1 in the time required to solve particular programming problems.
- More recent studies [Curtis 1981, DeMarco and Lister 1985, Brian 1997] confirm this high variability.
- Many employers in Silicon Valley argue that productivity variance is even higher today, perhaps as much as 100 to 1.

Variations in Programmer Productivity

- In 1968, a study by Sackman, Erikson, and Grant revealed that programmers with the same level of experience exhibit variations of more than 20 to 1 in the time required to solve particular programming problems.
- More recent studies [Curtis 1981, DeMarco and Lister 1985, Brian 1997] confirm this high variability.
- Many employers in Silicon Valley argue that productivity variance is even higher today, perhaps as much as 100 to 1.

Critical Observations about Software

1. Software development is an extraordinarily difficult task, exceeding in complexity most other engineering work. That difficulty, moreover, is intrinsic to the discipline and is not likely to change in the foreseeable future.

2. Software development requires people with an unusual combination of skills. Those people are in short supply, but their economic value is huge. Experienced programmers differ in productivity by several orders of magnitude.

3. Economic, social, and political factors are more important than technological progress in determining how computing evolves.

The Importance of Economics

Economics has more impact on directions in modern computing than technology does. The most significant factors are:

- *Low distribution costs.* Software is hugely expensive to produce, but essentially free to duplicate and distribute. Because development costs can be distributed across a larger base, big players have a distinct advantage.

The Importance of Economics

Economics has more impact on directions in modern computing than technology does. The most significant factors are:

- *Low distribution costs.* Software is hugely expensive to produce, but essentially free to duplicate and distribute. Because development costs can be distributed across a larger base, big players have a distinct advantage.
- *Network externalities.* The value of software increases with the number of people using that software.

The Importance of Economics

Economics has more impact on directions in modern computing than technology does. The most significant factors are:

- *Low distribution costs.* Software is hugely expensive to produce, but essentially free to duplicate and distribute. Because development costs can be distributed across a larger base, big players have a distinct advantage.
- *Network externalities.* The value of software increases with the number of people using that software.
- *Shortage of highly skilled labor.* The most productive programmers are in high demand, but short supply.

The Importance of Economics

Economics has more impact on directions in modern computing than technology does. The most significant factors are:

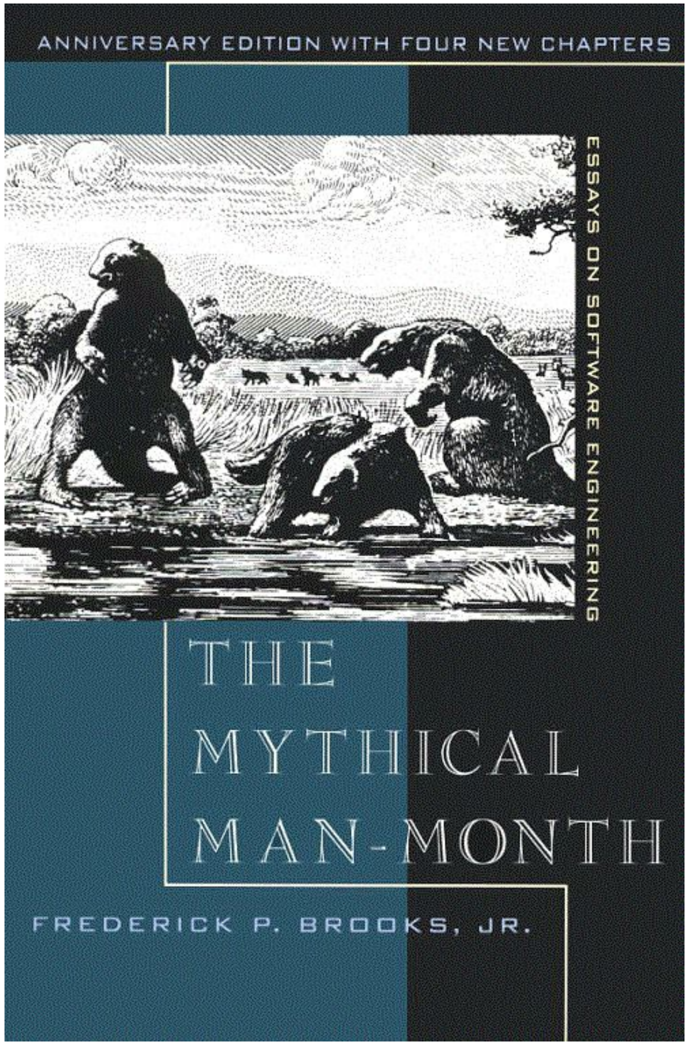
- *Low distribution costs.* Software is hugely expensive to produce, but essentially free to duplicate and distribute. Because development costs can be distributed across a larger base, big players have a distinct advantage.
- *Network externalities.* The value of software increases with the number of people using that software.
- *Shortage of highly skilled labor.* The most productive programmers are in high demand, but short supply.
- *High cost-effectiveness.* Software tends to be remarkably useful, even when bugs exist.

The Importance of Economics

Economics has more impact on directions in modern computing than technology does. The most significant factors are:

- *Low distribution costs.* Software is hugely expensive to produce, but essentially free to duplicate and distribute. Because development costs can be distributed across a larger base, big players have a distinct advantage.
- *Network externalities.* The value of software increases with the number of people using that software.
- *Shortage of highly skilled labor.* The most productive programmers are in high demand, but short supply.
- *High cost-effectiveness.* Software tends to be remarkably useful, even when bugs exist.

The Mythical Man-Month


ANNIVERSARY EDITION WITH FOUR NEW CHAPTERS
ESSAYS ON SOFTWARE ENGINEERING
THE MYTHICAL MAN-MONTH
FREDERICK P. BROOKS, JR.

Frederick P. Brooks, Jr., *The Mythical Man-Month: Essays on Software Engineering,* second edition, Reading, MA: Addison-Wesley, 1995.

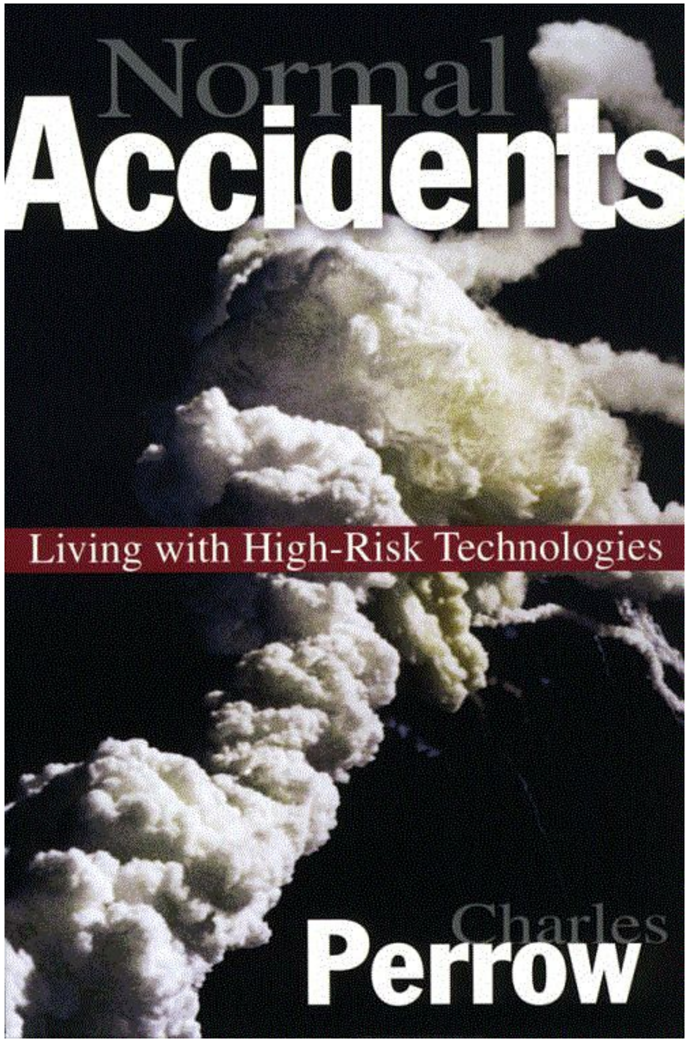
Originally published in 1975, *The Mythical Man-Month* remains the classic text on software engineering and its importance. Despite the fact that Brooks is an expert programmer with a background in both industry and academia, this book is easily accessible to a popular audience.

The Sachertorte Algorithm

John Shore, *The Sachertorte Algorithm and Other Antidotes to Computer Anxiety,* New York: Penguin Books, 1985.

This book is a highly accessible introduction to the complexities and pitfalls of software development.

Normal Accidents



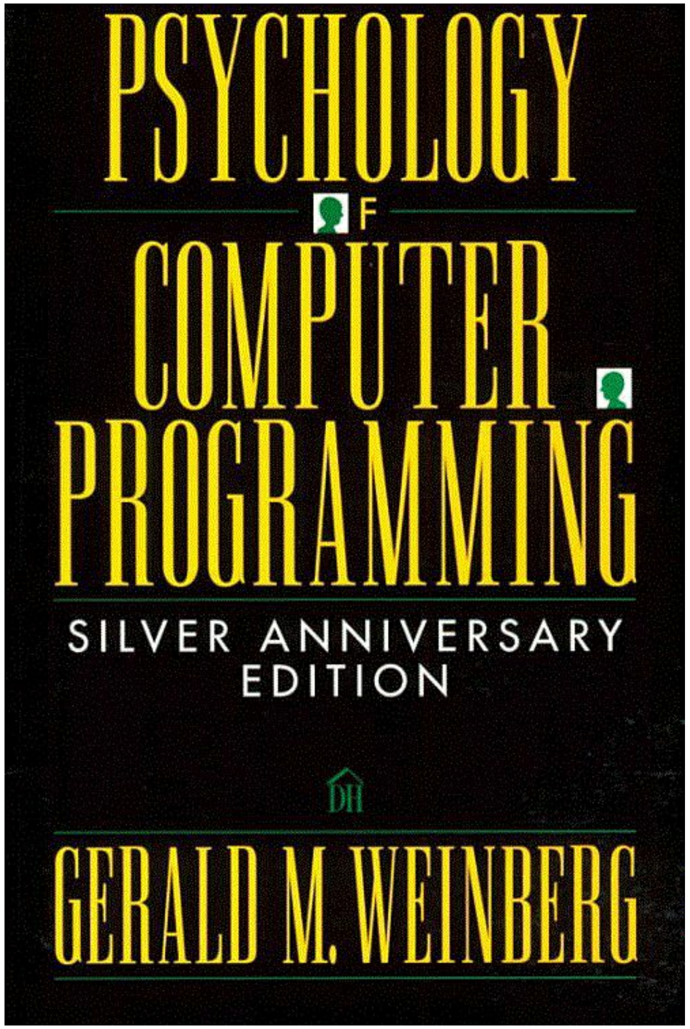
Charles Perrow, *Normal Accidents: Living with High-Risk Technologies*, New York: Basic Books, 1984.

Although this book does not focus specifically on programming—and indeed does not include software or programming in its index—the issues that it raises are critical to an understanding of why complex technological systems fail.

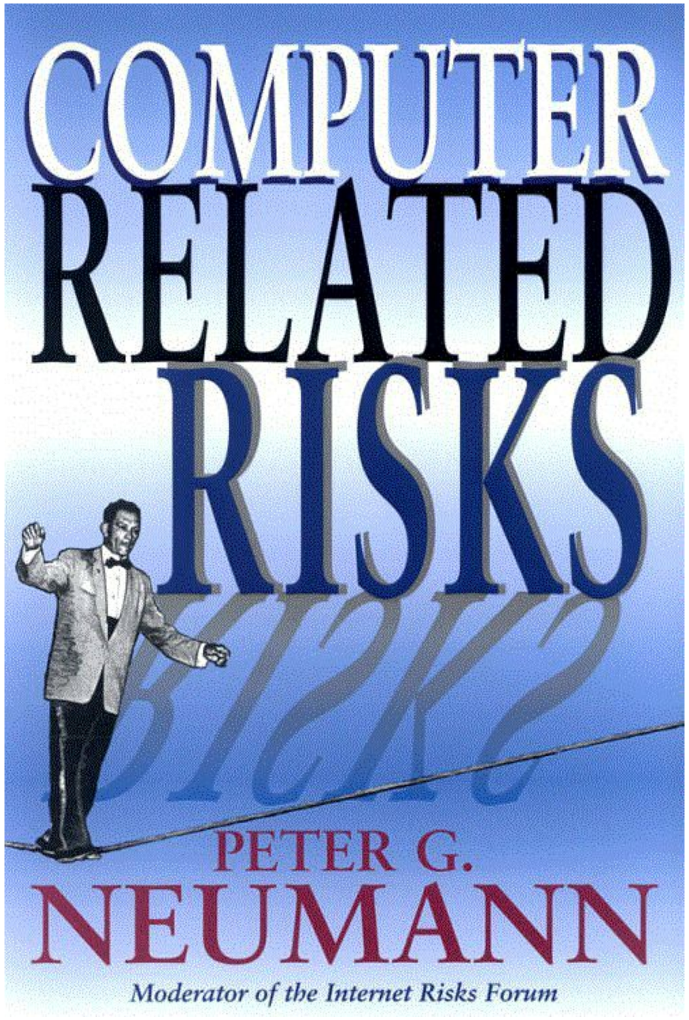
Psychology of Computer Programming

Gerald M. Weinberg, *Psychology of Computer Programming,* New York: Dorset House, 1998.

From the time it first appeared in 1971, this book has offered the best popular description of the mental working processes of programmers. The 1998 edition includes reflections on how well each chapter has held up over time.


PSYCHOLOGY OF COMPUTER PROGRAMMING
SILVER ANNIVERSARY EDITION
DH
GERALD M. WEINBERG

Computer Related Risks



Peter G. Neumann, *Computer Related Risks*, Reading, MA: Addison-Wesley, 1995.

This book presents a summary of the most compelling computer-related failures. Peter Neumann is best known as the moderator of the Internet Risks Forum available at

`http://catless.ncl.ac.uk/Risks/`