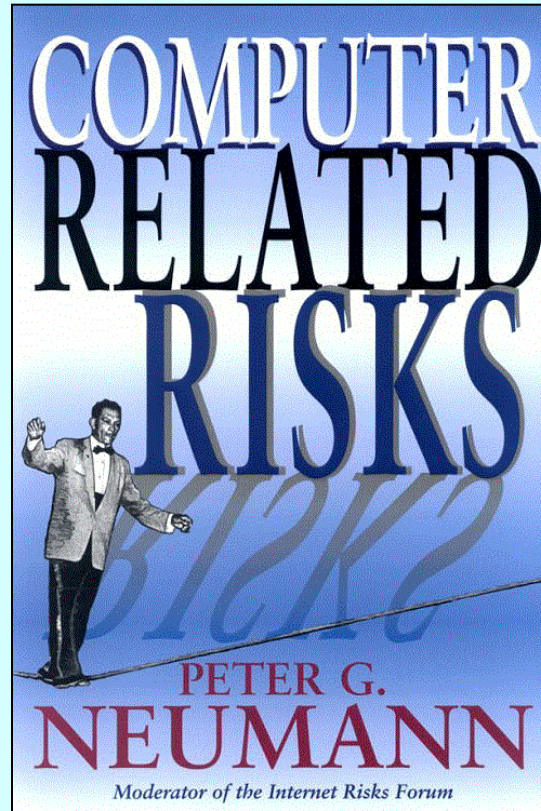


# Reliability and Risk



# Computing Ethics in the news

- China has the world's fastest supercomputer
  - <http://www.nytimes.com/2016/06/21/technology/china-tops-list-of-fastest-computers-again.html>
  - <http://www.nytimes.com/2016/06/22/technology/daily-report-china-has-the-worlds-most-powerful-computer-again.html>
  - In the US, the government has been moving away from funding basic scientific research, and companies "like Google, Microsoft, Facebook and Amazon invested billions of dollars in cloud-computing centers that don't focus as much on solving scientific problems."

# AI's "White guy" problem

- <http://www.nytimes.com/2016/06/26/opinion/sunday/artificial-intelligences-white-guy-problem.html>
- <https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing>
- <http://www.cmu.edu/news/stories/archives/2015/july/online-ads-research.html>
- "Sexism, racism and other forms of discrimination are being built into the machine-learning algorithms that underlie the technology behind many 'intelligent' systems that shape how we are categorized and advertised to."

# AI's “White guy” Problem

- Google's photo app was classifying pictures of black people as gorillas
- Nikon camera misreading pictures of Asian people as blinking
- HP camera software had troubles identifying dark-skinned people
- Amazon same-day delivery service was not available to people in many heavily black neighborhoods
- Propublica reported that blacks are twice as likely to re-offend
- “Predictive policing” could pose a similar danger
- Women less likely to be shown ads for high-paying jobs

# Tech and the SF housing crisis

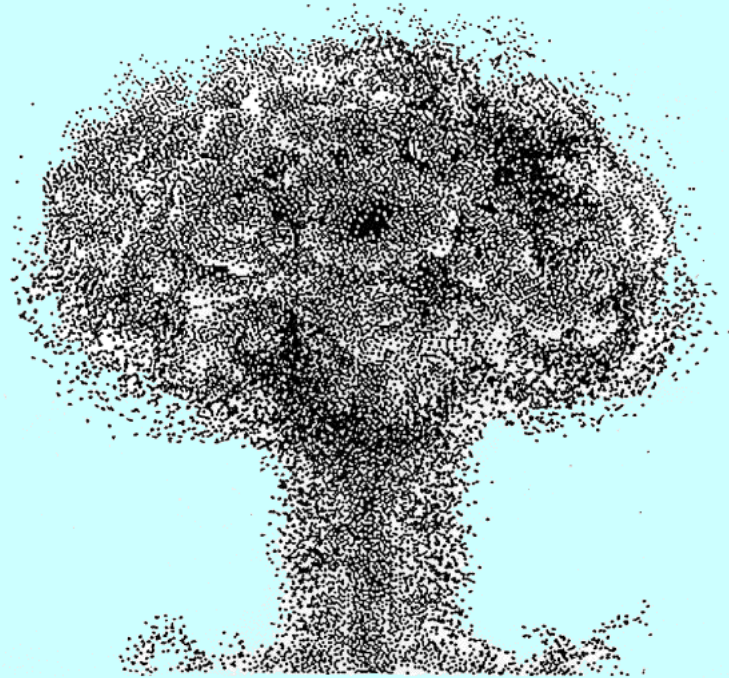
- <http://techcrunch.com/2014/04/14/sf-housing/>
- <http://techcrunch.com/2015/01/10/east-of-palo-altos-edden/>
  - Gentrification and the "Great Inversion"
  - Displacement of the poor and lower middle class communities
    - While this topic is focused on SF/SV, there are obvious global implications as well – if this can happen here, it can happen anywhere

# 23andMe

- <http://gizmodo.com/of-course-23andmes-business-plan-has-been-to-sell-your-1677810999>
  - "Genentech will pay up to \$60 million for access to 23andMe's data to study Parkinson's"
- <http://www.scientificamerican.com/article/23andme-is-terrifying-but-not-for-the-reasons-the-fda-thinks/>
  - Serious “big data implications
  - Another interesting final presentation topic!

# Part I

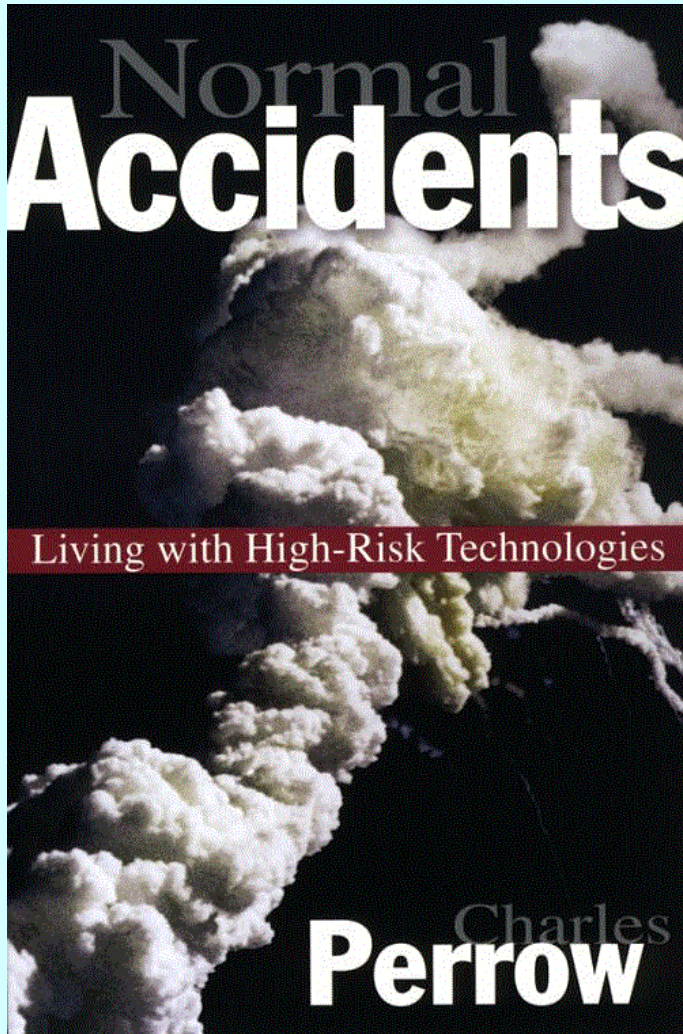
## The Nature of Software Failure



**THE ULTIMATE ERROR MESSAGE**  
Computer Professionals for Social Responsibility



# Normal Accidents

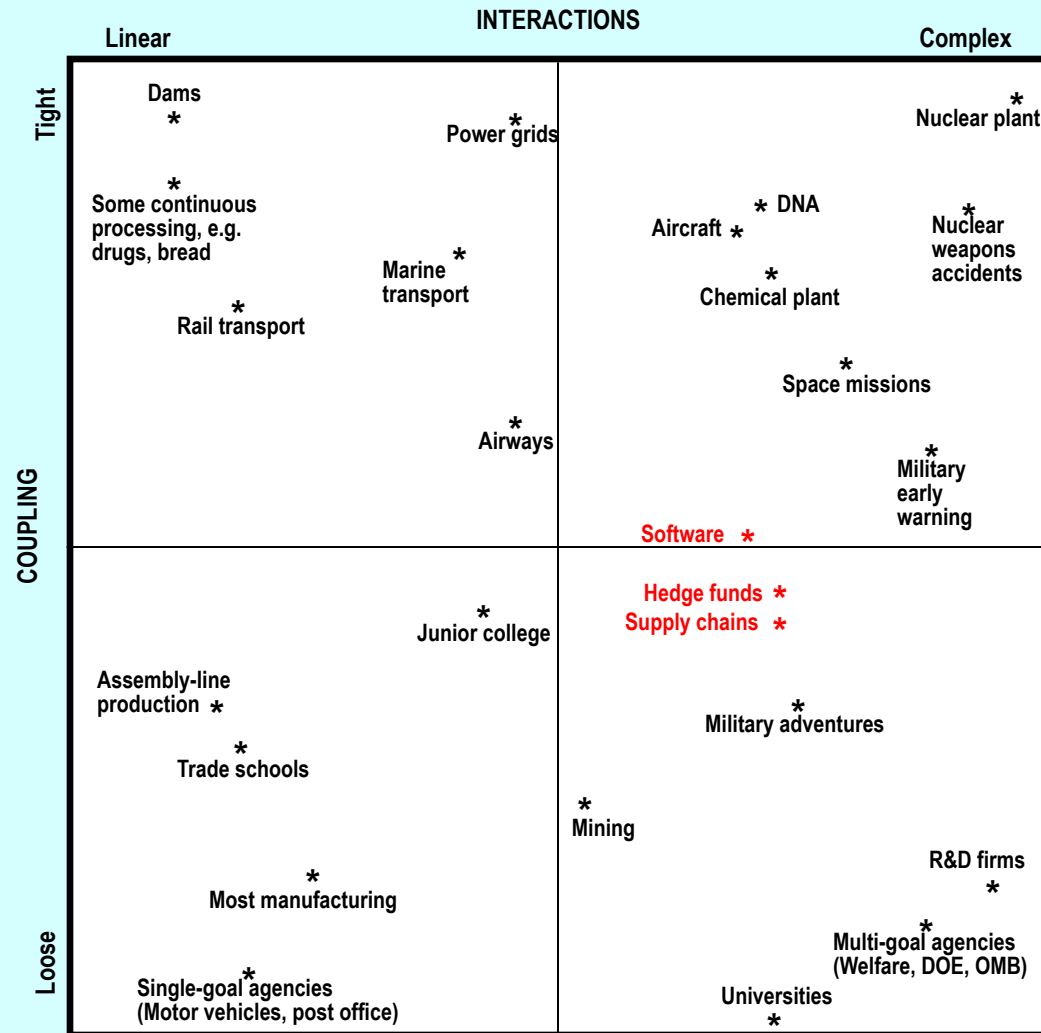


Charles Perrow, *Normal Accidents: Living with High-Risk Technologies*, New York: Basic Books, 1984.

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



# Coupling and Interaction Level



# 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. Individual software developers differ remarkably in their level of productivity, sometimes by several orders of magnitude. Very good people are in short supply, even though their economic value is huge. Unfortunately, the shortfall appears to be getting worse rather than better.
3. The direction of evolution in computing is not controlled solely by technological innovation. Economic, social, and political factors play at least as large a role.

# The Discovery of Debugging

9/9


0800 Anttan started  
1000 " stopped - anttan ✓

1300 (033) HP - MC { 1.2700 9.037847025  
2.130476415 (2) 4.615925059(-2)  
033) PRO 2 2.130476415  
convd 2.130676415

Relays 6-2 in 033 failed special speed test  
in relay 11.000 test.

Relays changed

1100 Started Cosine Tape (Sine check)  
1525 Started Multi-Adder Test.

1545  Relay #70 Panel F  
(moth) in relay.

First actual case of bug being found.

1630 unchanging started.  
1700 closed down.

—Grace Murray Hopper's notebook  
Mark I project at Harvard University  
September 9, 1947

# The Discovery of Debugging



Maurice Wilkes  
1913-2010

“As soon as we started programming, we found to our surprise that it wasn’t as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs.”

—Maurice Wilkes, lecture on “The Design and Use of the EDSAC,” September 1979

# Bugs are Unavoidable

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

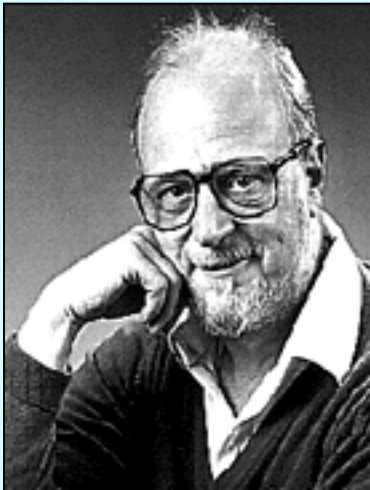
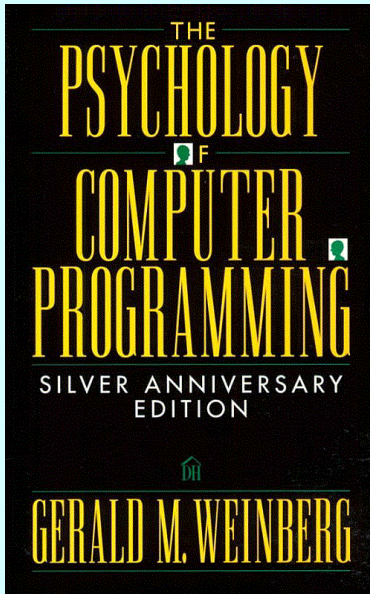
# The Difficulty of Software

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 of Strategic Defense Systems*, 1985



# The Complexity of Debugging



Gerald Weinberg, 1933-

First, there is only the *gestalt*, a general feeling that something is out of place without any particular localization. Then follows the ability to shake loose from an unyielding situation—the ability to change one's point of view. . . . Then, however, one must go from the general to the particular—"focusing" as it was called here. Although one does not find errors by a detailed search of each line, word, or character, the ability to get down to details is essential in the end. Thus, for debugging, an almost comple-mentary set of mental powers is needed. No wonder good debuggers are so rare!

—Gerald Weinberg, *The Psychology of Computer Programming*, 1971

# No Silver Bullet

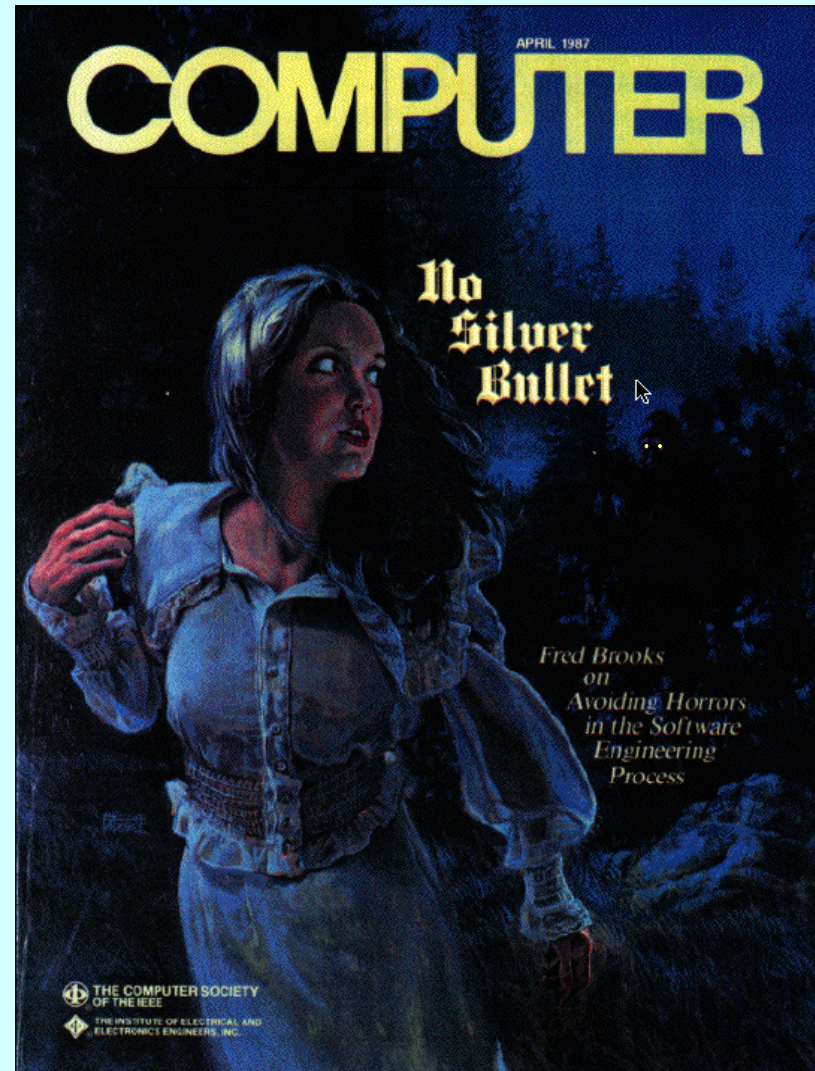
- What are accidental difficulties?
- What are essential difficulties?
- Why did I ask you to read this article?

# Essential and Accidental Complexity

To see what rate of progress one can expect in software technology, let us examine the difficulties of that technology. Following Aristotle, I divide them into *essence*, the difficulties inherent in the nature of software, and *accidents*, those difficulties that today attend its production but are not inherent. . . .

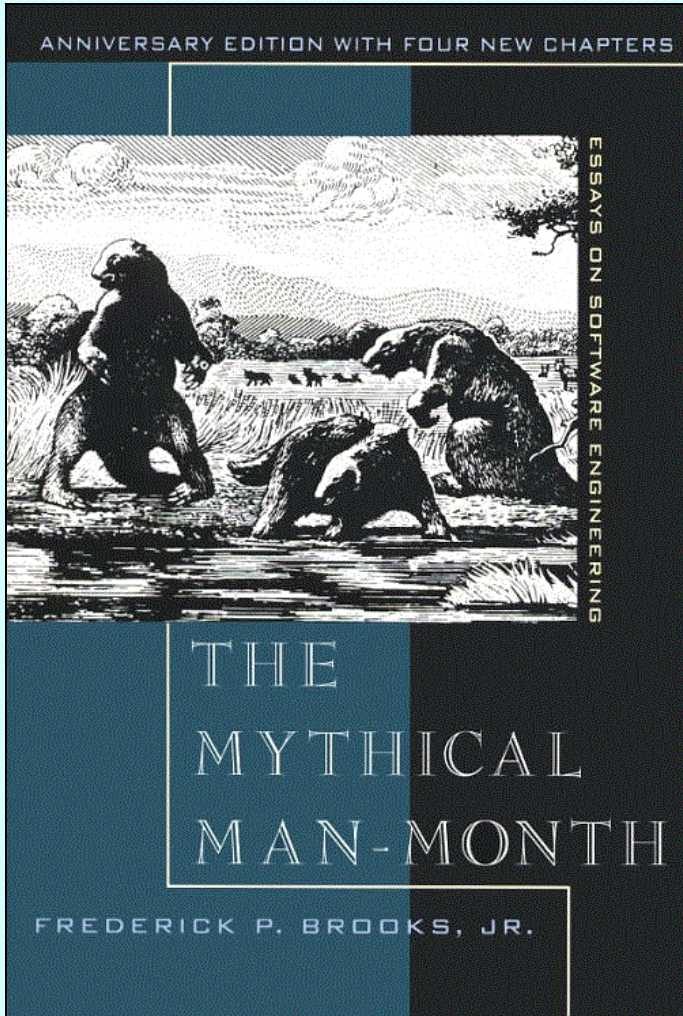
The complexity of software is an essential property not an accidental one. Hence, descriptions of a software entity that abstract away its complexity often abstract away its essence.

—Fred Brooks  
“No Silver Bullet”  
*IEEE Computer*, April 1987





# The Mythical Man-Month



Frederick 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 a classic text on software engineering and its importance. Despite the fact that Brooks is an expert programmer with experience in both industry and academia, this book is accessible to a broad audience.

# Brooks' s Law

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

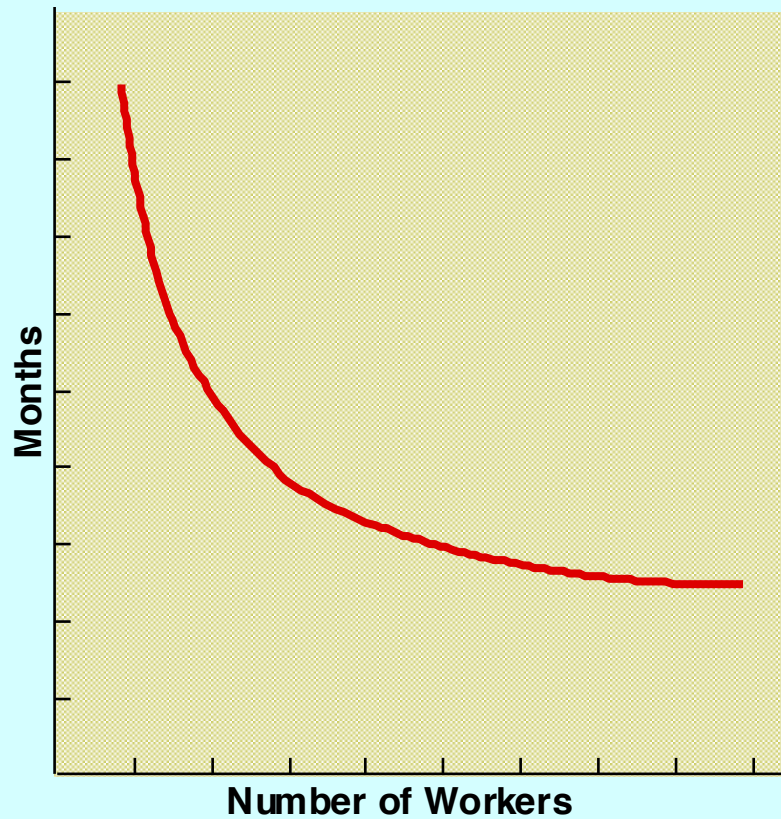


Fig. 2.3 Time versus number of workers—  
partitionable task requiring communication

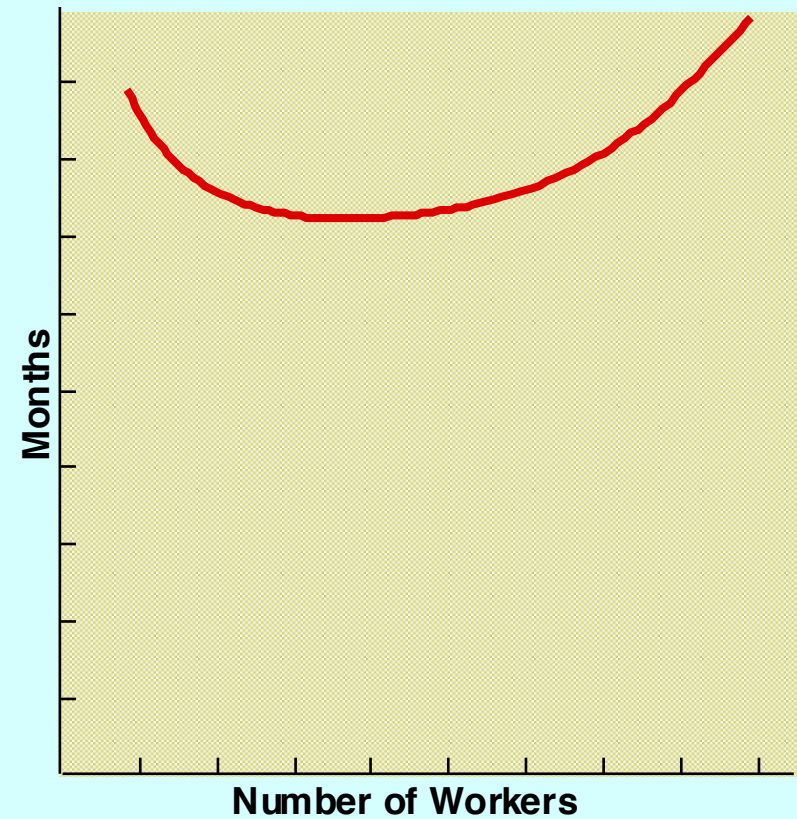


Fig. 2.4 Time versus number of workers—  
task with complex interrelationships

# One last note on software complexity

- Intel' s “Parallel Gamble” is likely going to make things worse from:
  - An educational perspective
  - A practical perspective



# 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. Individual software developers differ remarkably in their level of productivity, sometimes by several orders of magnitude. Very good people are in short supply, even though their economic value is huge. Unfortunately, the shortfall appears to be getting worse rather than better.
3. The direction of evolution in computing is not controlled solely by technological innovation. Economic, social, and political factors play at least as large a role.

# 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, with estimates sometimes exceeding two orders of magnitude. In 2006, Alan Eustace, Google's Vice President of Engineering, told *The Wall Street Journal* that the best Google engineers were 300 times more productive than the average.

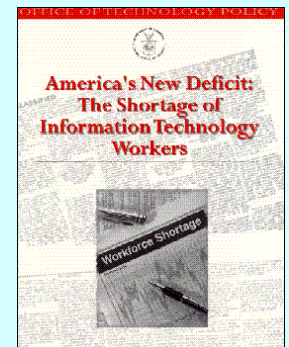
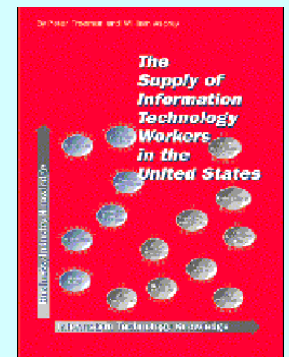
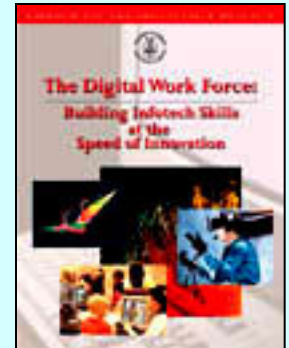
# Critical Observations about Software

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. Individual software developers differ remarkably in their level of productivity, sometimes by several orders of magnitude. Very good people are in short supply, even though their economic value is huge. Unfortunately, the shortfall appears to be getting worse rather than better.
3. The direction of evolution in computing is not controlled solely by technological innovation. Economic, social, and political factors play at least as large a role.

# The Labor Shortage in Computing

For many years, computing industry insiders have argued that a significant labor shortage exists in certain specialty areas—primarily those associated with software development—within the broader sphere of Information Technology, even after the economic downturn associated with the collapse of the Internet bubble.

This position, however, has been difficult to prove, largely because the data available from the Bureau of Labor Statistics is aggregated in such a way that shortages in specific skill classes are difficult to recognize. The controversy has intensified as some analysts offer alternative explanations for problems reported by the industry.



# Top Ten Fastest Growing Jobs

	Job category	Employment (thousands)		Growth
		2006	2016	
1.	Network systems and data communications analysts	262	402	53.4
2.	Personal and home care aides	767	1,156	50.6
3.	Home health aides	787	1,171	48.7
4.	Computer software engineers, applications	507	733	44.6
5.	Veterinary technologists and technicians	71	100	41.0
6.	Personal financial advisors	176	248	41.0
7.	Makeup artists, theatrical and performance	2	3	39.8
8.	Medical assistants	465	148	35.4
9.	Veterinarians	62	84	35.0
10.	Substance abuse and behavioral disorder counselors	83	112	34.3

Source: U.S. Department of Labor, Bureau of Labor Statistics, *Employment Projections: 2006-16*, December 2007.

# The Importance of Economics

Economics often has more impact on the direction of computing evolution 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.



# Part II

## Case Studies

# Summary of the Therac-25 Case

- Two companies—the Atomic Energy Commission Limited (AECL) of Canada and CGR of France—collaborated to build a series of radiation-therapy devices called *Therac*.
- Before shipping the first Therac-25 in 1983, AECL had built two previous models: the Therac-6 and the Therac-20.
- Eleven Therac-25s were installed in the U.S. and Canada.
- From 1985 to 1987, Therac-25 units were involved in six incidents of accidental overdose. Three deaths were directly attributable to the overdose; a fourth patient received a life-threatening overdose but died from the original cancer.
- Throughout this period, AECL discounted the possibility of software failures in their machine. The problems were identified by a physicist at the installation in Tyler, Texas.

# Timeline of the Therac-25 Case

- 1983**            July    AECL introduces the new Therac-25 machine.
  
- 1985**            June 3    Kennestone Regional Oncology Center, Marietta Georgia. Linda Knight is overdosed. Kennestone physicist asks AECL if overdose is possible. He is informed it is not.
  
- July 26    Hamilton, Ontario. Donna Gartner is overdosed. AECL sends service engineer to investigate.
  
- November 8    CRPB requests that AECL implement hardware safety interlocks.
  
- November 18    Linda Knight files suit against AECL.
  
- December    Yakima Valley Memorial Hospital, Yakima, Washington. Janis Tilman is overdosed.

# Timeline of the Therac-25 Case

- 1986**
- April 11 ETCC. Daniel McCarthy is overdosed. ETCC physicist discovers the cause of “Malfunction 54.” He explains findings to AECL.
  - April 14 AECL files first report with United States FDA. AECL sends letters to Therac-25 users with suggestions to help avoid future accidents.
  - May 1 Daniel McCarthy dies from radiation burns.
  - May 2 FDA declares Therac-25 defective and demands a Corrective Action Plan from AECL.
  - August First Therac-25 users group meeting.
  - August Isaac Dahl dies as a result of radiation poisoning.

# Timeline of the Therac-25 Case

- 1987** January 11 Yakima Valley Memorial Hospital. Anders Engman overdosed.
- January 19 AECL issues hazard notification to Therac-25 users.
- February 10 FDA sends notice to AECL advising that Therac-25 is defective under U.S. law, and requests that Therac-25 be taken out of use.
- March Second user group meeting.
- April Anders Engman dies of complication from radiation burns.
- July Third users group meeting.
- 1988** November Final Safety Analysis Report issued.

# Therac-25: Contributing Factors

- Management inadequacies and lack of procedure for following through on all reported incidents.
- Overconfidence in the software and removal of hardware interlocks, making the software into a single point of failure that could lead to an accident.
- Presumably less-than-acceptable software-engineering practices.
- Unrealistic risk assessments along with overconfidence in the results of those assessments.



# Therac-25: Lessons Learned

- Documentation should not be an afterthought.
- Software quality assurance practices and standards should be established.
- Designs should be kept simple.
- Ways to get information about errors—for example, software audit trails—should be established.
- The software should be subjected to extensive testing and formal analysis at the module and system level; system testing alone is not adequate.
- Safety must be built into software, and, in addition, safety must be assured at the system level despite software errors.
- Reusing software modules does not guarantee safety in the new system to which they are transferred and sometimes leads to awkward and dangerous designs.

# Report Conclusion

Fixing each individual software flaw as it was found did not solve the device's safety problems.

Virtually all complex software will behave in an unexpected or undesired fashion under some conditions—there will always be another bug.

Instead, accidents must be understood with respect to the complex factors involved. In addition, changes need to be made to eliminate or reduce the underlying causes and contributing factors that increase the likelihood of accidents or loss resulting from them.

# And from an information security perspective

- Integrity – The radiation dosage was altered by the system

# Case Study 2: The Ariane 5

- The problem: 37 seconds after liftoff, the Ariane 5 rocket nosedived, and crashed

# Cause

- Reuse of the Inertial Reference System (SRI) from the Ariane 4
- The internal SRI software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer. This resulted in an Operand Error. The data conversion instructions (in Ada code) were not protected from causing an Operand Error, although other conversions of comparable variables in the same place in the code were protected.

# Recommendations

- Testing
- Challenges with reuse
- Handling of catastrophic failure (especially from single points of failure)
- Better SE process (especially reviews)
- “A more transparent organisation of the cooperation among the partners in the Ariane 5 programme must be considered. Close engineering cooperation, with clear cut authority and responsibility, is needed to achieve system coherence, with simple and clear interfaces between partners.”



# Information Security issues