

THE COMPUTING UNIVERSE

TONY HEY AND GYURI PÁPAY

A JOURNEY THROUGH A REVOLUTION

The Computing Universe

A Journey through a Revolution

Computers now impact almost every aspect of our lives, from our social interactions to the safety and performance of our cars. How did this happen in such a short time? And this is just the beginning....

In this book, Tony Hey and Gyuri Pápay lead us on a journey from the early days of computers in the 1930s to the cutting-edge research of the present day that will shape computing in the coming decades. Along the way, they explain the ideas behind hardware, software, algorithms, Moore's law, the birth of the personal computer, the Internet and the web, the Turing Test, IBM's *Jeopardy!*-beating Watson, World of Warcraft, spyware, Google, Facebook, and quantum computing. This book also introduces the fascinating cast of dreamers and inventors who brought these great technological developments into every corner of the modern world.

This exciting and accessible introduction will open up the universe of computing to anyone who has ever wondered where his or her smart phone came from and where we may be going in the future.



TONY HEY is vice president of Microsoft Research and has published more than 150 research papers and several books, including *Gauge Theories in Particle Physics* (fourth edition, 2012, with Ian Aitchison), *The New Quantum Universe* (second edition, 2003, with Patrick Walters), *Einstein's Mirror* (1997, with Patrick Walters), and *The Fourth Paradigm* (2009, edited with Stewart Tansley and Kristin Tolle).



GYURI PÁPAY is a senior research Fellow at the IT Innovation Centre at Southampton University, U.K. His main research interests are high-performance computing, event simulations, and numerical algorithms.

The **Computing Universe**

A Journey through a Revolution

Tony Hey

Microsoft Research, Redmond, Washington

Gyuri Pápay

IT Innovation Centre, Southampton, U.K.



CAMBRIDGE
UNIVERSITY PRESS



32 Avenue of the Americas, New York, NY 10013-2473, USA

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning, and research at the highest international levels of excellence.

www.cambridge.org

Information on this title: www.cambridge.org/9780521150187

© Tony Hey and Gyuri Pápay 2015

This publication is in copyright. Subject to statutory exception and to the provisions of relevant collective licensing agreements, no reproduction of any part may take place without the written permission of Cambridge University Press.

First published 2015

Printed in the United States of America

A catalog record for this publication is available from the British Library.

Library of Congress Cataloging in Publication data
Hey, Anthony J. G.

The computing universe : a journey through a revolution / Tony Hey, Microsoft Research, Redmond, Washington, Gyuri Pápay, IT Innovation Centre.

pages cm

Includes bibliographical references and index.

ISBN 978-0-521-76645-6 (hardback) – ISBN 978-0-521-15018-7 (pbk.)

1. Computer science – History. I. Pápay, Gyuri. II. Title.

QA76.17.H49 2014

004.09–dc23 2014030815

ISBN 978-0-521-76645-6 Hardback

ISBN 978-0-521-15018-7 Paperback

Additional resources for this publication at www.thecomputinguniverse.org

Cambridge University Press has no responsibility for the persistence or accuracy of URLs for external or third-party Internet web sites referred to in this publication and does not guarantee that any content on such web sites is, or will remain, accurate or appropriate.

Contents

	<i>page</i>
Preface	vii
A quick tour through the book	xi
Acknowledgments	xiii
Prologue: Blog entry from Jonathan Hey	xv
1 Beginnings of a revolution	1
2 The hardware	23
3 The software is in the holes	39
4 Programming languages and software engineering	58
5 Algorithmics	84
6 Mr. Turing's amazing machines	102
7 Moore's law and the silicon revolution	120
8 Computing gets personal	141
9 Computer games	174
10 Licklider's Intergalactic Computer Network	192
11 Weaving the World Wide Web	220
12 The dark side of the web	243
13 Artificial intelligence and neural networks	263
14 Machine learning and natural language processing	280

Contents

15 The end of Moore's law	298
16 The third age of computing	318
17 Computers and science fiction - an essay	333
Epilogue: From Turing's padlocked mug to the present day	359
Appendix 1. Length scales	361
Appendix 2. Computer science research and the information technology industry	362
How to read this book	365
Notes	367
Suggested reading	377
Figure credits	381
Name index	389
General index	393

Preface

Inspirations

There are many “popular” books on science that provide accessible accounts of the recent developments of modern science for the general reader. However, there are very few popular books about computer science – arguably the “science” that has changed the world the most in the last half century. This book is an attempt to address this imbalance and to present an accessible account of the origins and foundations of computer science. In brief, the goal of this book is to explain how computers work, how we arrived at where we are now, and where we are likely to be going in the future.

The key inspiration for writing this book came from Physics Nobel Prize recipient Richard Feynman. In his lifetime, Feynman was one of the few physicists well known to a more general public. There were three main reasons for this recognition. First, there were some wonderful British television programs of Feynman talking about his love for physics. Second, there was his best-selling book *“Surely You’re Joking, Mr. Feynman!”: Adventures of a Curious Character*, an entertaining collection of stories about his life in physics – from his experiences at Los Alamos and the Manhattan atomic bomb project, to his days as a professor at Cornell and Caltech. And third, when he was battling the cancer that eventually took his life, was his participation in the enquiry following the *Challenger* space shuttle disaster. His live demonstration, at a televised press conference, of the effects of freezing water on the rubber O-rings of the space shuttle booster rockets was a wonderfully understandable explanation of the origin of the disaster.

Among physicists, Feynman is probably best known for Feynman diagrams, the work that brought him his Nobel Prize in 1964. The diagrams constitute a calculational tool kit that enables physicists to make sense of not only Quantum Electrodynamics, the theory that underpins electricity and magnetism, but also of the relativistic quantum field theories believed to describe the weak and strong interactions of elementary particles. But Feynman was not only a great researcher: his *Feynman Lectures on Physics* are a masterly three-volume introduction to modern physics based on lectures that he gave at Caltech in the 1960s. He was also a visionary: Feynman’s after-dinner talk “There’s Plenty of Room at the Bottom” in 1959 first introduced the ideas of nanotechnology – the behavior of devices at length scales approaching atomic dimensions.

Preface

By the early 1980s, Feynman had become interested in computing, and for the last five years of his life, he gave a lecture course on computing. In the first two years, he collaborated on an ambitious computing course with two Caltech colleagues, Carver Mead and John Hopfield. In the third year, assisted by Gerry Sussman, a computer scientist on sabbatical from MIT, Feynman gave his own version of the course. The lectures contained a fascinating mixture of standard computer science material plus discussion of the thermodynamics of computing and an analysis of a quantum computer. Before he died, Feynman asked Tony Hey to edit his notes for publication, and the lectures eventually saw the light of day as *The Feynman Lectures on Computation*. Feynman also acted as a consultant to the Thinking Machines computer company, founded by MIT researcher Danny Hillis.

Feynman's introductory lectures on quantum mechanics were the inspiration for *The New Quantum Universe*, an earlier popular science book by one of the present authors. Feynman's computing lectures seemed to invite the creation of a similar popular treatment. Feynman had also given an introductory talk on computers in his "Computers from the Inside Out" lecture at the Esalen Institute – a "holistic learning and retreat center" – in Big Sur, California. In this talk he explained the essential working of a computer using the analogy of a "very dumb file clerk." Thus Feynman's lectures on computing were the original inspiration for our attempt at a popular book on computer science.

There were several other sources of inspiration for this book. One was *The Soul of a New Machine* by Tracy Kidder. Although that book is about the design and construction of a new mini-computer, it reads like a thriller – and even has a chapter titled "The Case of the Missing NAND Gate." Another inspiration was the book *The State of the Art*, a pictorial history of Moore's law by computer historian Stan Augarten. It is another book by Augarten, *Bit by Bit* – an illustrated history of computers, from calculating machines to the personal computer – that is closest in spirit to the present book. Other inspirations were *Algorithmics* by the Israeli computer scientist David Harel, originally given as a series of radio lectures, and *The Pattern in the Stone* by the computer architect Danny Hillis.

Digital literacy and computer science

At school, we take proficiency in the "3 Rs" – reading, writing, and arithmetic – to be an essential life skill. Now, in addition, we expect that all children should know how to use computers – to produce electronic documents, manipulate spreadsheets, make slide-show presentations, and browse the web. But such basic "digital literacy" skills are not what is meant by the term *computer science*. Computer science is the study of computers – how to build them, understand their limitations, and use their power to solve complex problems. Alan Turing, the English genius who was one of the first to explore these questions, developed a theoretical machine model by imitating how a "human computer" would go about solving a computational problem. Turing machines provide the essential mathematical basis for reasoning about the behavior of computers. But computer science is about more than mathematics; it is also about engineering – building complex systems that do useful things. In computer

engineering we also have the additional freedom to explore virtual systems – complex structures without the limitations of real physical systems.

Computer scientist Jeannette Wing defines *computational thinking* as the ability to use the fundamental concepts of computer science to solve difficult problems, design complex systems, and understand human behavior. She believes that education in computational thinking will be as essential in the twenty-first century as the 3 Rs have been in all previous centuries. Computational thinking includes techniques of abstraction and decomposition that assist in the development of algorithms to attack complex tasks or to design complex systems. It also gives new insights on system concepts such as prevention, protection, and recovery by thinking in terms of corresponding computer science concepts such as redundancy, damage containment, and error correction. Computational thinking can also help apply ideas from machine learning and Bayesian statistics to everyday problems. In many areas of life we are faced with the problem of planning and learning in the presence of uncertainty, and computational thinking ideas applied to “big data” have application in both science and commerce.

How do we instruct a computer to solve a particular problem? First we must write down our *algorithm* – a sequence of steps to the solution rather like a cooking recipe – in a specialized *programming language*. The specific sequence of instructions is called a *program* and this constitutes part of the *computer software* required to solve the problem on the computer. The programming language instructions can then be translated into operations that can be carried out by the low-level components of the *computer hardware*. Running the program requires more software, the *operating system* that manages the input and output of data and the use of storage devices and printers. Programming is the skill required to translate our computer science algorithms into programs that computers can understand. Like digital literacy skills, the ability to program is certainly a vital skill for a future career in the information technology industry but constitutes only a small part of Jeannette Wing’s computational thinking agenda.

The goal of this book

Our goal in writing this book is not to produce another textbook on computers, or a book on the history of computing. Rather, the book is intended to be intelligible to both high school and first-year university students and to stimulate their interest in the wide range of opportunities of a career in computer science. We also hope that it will provide general readers with an understandable and accessible account of how computers work, as well as a glimpse of the vast scope of activities that have been enabled by networks of interconnected computers. In order to make the book more readable and entertaining we have included brief biographies and anecdotes about the scientists and engineers who helped create this computing universe.

It is curious that schoolchildren are taught the names and achievements of great mathematicians, physicists, chemists, and biologists but not about the great computer pioneers. In part then, one goal of the book is to make a start at correcting this imbalance by highlighting the contributions of the pioneers

Preface

of computing. These include the early theoretical ideas of Alan Turing and John von Neumann as well as the achievements of the first computer engineers such as Presper Ekert and John Mauchly in the United States and Maurice Wilkes and Konrad Zuse in Europe. The story follows the rise of IBM and Digital to the computing legends at Xerox PARC with their incredible Alto computer, created by Alan Kay, Chuck Thacker, and Butler Lampson. In a very real sense, the story of computing follows the evolution of Moore's law and the rise of the semiconductor industry. It was the microprocessor – "a computer on a chip" – that made possible the personal computing revolution with pioneers like Steve Jobs and Steve Wozniak from Apple and Bill Gates and Paul Allen from Microsoft.

If the first thirty years of computers were about using computers for computing, the second thirty years have been about using computers for communicating. The story takes us from the earliest speculations about interactive computing and the Internet by J. C. R. Licklider; to the packet-switching ideas of Paul Baran and Donald Davies; to the ARPANET of Bob Taylor, Larry Roberts, and BBN; to the Internet protocols of Bob Kahn and Vint Cerf. Early ideas about hypertext and linked documents of Vannevar Bush, Ted Nelson, and Doug Engelbart evolved into the now ubiquitous World Wide Web, created by Tim Berners-Lee. Similarly, the PageRank algorithm, invented by Stanford graduate students Sergey Brin and Larry Page, led to the rise of Internet search engines like Google, Bing, and Baidu.

Today, we are able to forecast the weather with reasonable accuracy; access vast amounts of information; talk to anybody over the Internet; play games, collaborate, and share information easily with others; and, if we wish, broadcast our thoughts to the entire world. Already, the opportunities of our present computing universe seem endless, yet we are only at the beginning of what will be possible in the future. According to Turing Award recipient Butler Lampson, the next thirty years will see us enter the *Third Age of Computing* in which computers become able to act intelligently on our behalf. These developments will bring with them serious issues concerning ethics, security, and privacy, which are beyond the scope of this book. Instead, the book ends with a look at some possible computing technologies and computer science challenges for the future.

A quick tour through the book

[Chapters 1 to 6](#) take the reader from the beginnings of digital computers to a description of how computer hardware and software work together to solve a problem. The ideas of programming languages and software engineering are covered in [Chapter 4](#), and computer algorithms in [Chapter 5](#). [Chapter 6](#) is probably the most difficult chapter in the book as it tries to explain the fundamental theoretical insights of Alan Turing and Alonzo Church on computability and universality. This chapter can be skipped on a first reading without jeopardizing the understandability of the later chapters. For readers interested in hardware, [Chapter 7](#) contains accounts of the discovery of the transistor and the integrated circuit or silicon chip and the origins of Moore's law, as well the quantum mechanics of semiconductors. [Chapter 15](#) looks at the coming end of Moore's law and some future alternatives to silicon as the miniaturization level approaches atomic dimensions.

The history sections at the ends of [Chapters 1](#) and [2](#) offer more background in the history of computer science, including the very early ideas of Charles Babbage and Ada Lovelace; the little-known Colossus computer, developed at the UK Post Office's Dollis Hill research laboratory for use by the code breakers at Bletchley Park; LEO, the first business computer; and the first stored-program computers, the Manchester Baby and the Cambridge EDSAC. In [Chapter 8](#) there is also a history section describing the pioneers of interactive and personal computing.

[Chapter 8](#) describes the development of personal computers based around microprocessors and the key roles played by Xerox PARC, IBM, Microsoft, and Apple in moving to the present era of smart phones, tablets, and touch interfaces. [Chapter 9](#) describes the origins of computer games and computer graphics. The three key chapters about the Internet, World Wide Web, search engines, and malware are [Chapters 10, 11, and 12](#).

Artificial intelligence and the famous Turing Test are the subject of [Chapter 13](#), while [Chapter 14](#) describes modern applications of machine learning technologies to computer vision, speech, and language processing. All of these things were involved in the design of IBM's Watson machine that won on the TV game show *Jeopardy!*. [Chapter 16](#) looks to the future with an account of progress in robotics and the coming Internet of Things. The chapter ends with a discussion of Strong AI and the problem of consciousness.

[Chapter 17](#) is an essay about computers in science fiction.

More detailed advice about ways to read this book is included at the end of the book.

Acknowledgments

Unsurprisingly perhaps, this project has taken much longer than we ever envisioned. Our families certainly deserve our unreserved thanks and gratitude for their support and forbearance over the five years and more it has taken us to get the project to completion. Tony Hey thanks his wife Jessie; children Nancy, Jonathan, and Christopher; and son-in-law Jonathan Hoare and daughter-in-law Maria Hey. Gyuri thanks his wife Ivetka and daughter Mónika.

Tony particularly wants to thank his colleagues at Microsoft for their helpful input and corrections: Gordon Bell, Doug Berger, Judith Bishop, Barry Briggs, Bill Buxton, T. J. Campana and the Microsoft Digital Crimes Unit, Scott Charney, Li Deng, Andrew Fitzgibbon, Katy Halliday, Jeff Han, David Heckerman, Carl Kadie, Kevin Kutz and his sons Michael and Joe, Brian LaMacchia, Butler Lampson, Peter Lee, Roy Levin, Qi Lu, Nachi Nagappan, Savas Parastatidis, Jim Pinkelman, Mark Russinovich, Jamie Shotton, Amy Stevenson, Krysta Svore, Chuck Thacker, Evelyne Viegas, and Jeannette Wing. He also thanks Craig Mundie, Bill Gates, Rick Rashid, and Steve Ballmer for the opportunity to join Microsoft Research and work in a great IT company.

Gyuri wishes to thank all his colleagues at the IT Innovation Centre at the University of Southampton. In particular he would like to thank Colin Upstill, Mike Surridge, Michael Boniface, and Paul Walland for their help, advice, and support.

We also wish to thank Hal Abelson, Gary Alt, Martin Campbell-Kelly, Sue Carroll and Apple, Sara Dreyfuss, George Dyson, Amy Friedlander, Wendy Hall, David Harel, Tony Hoare, John Hollar and the Computer History Museum, John Hopcroft, Scott Howlett and IBM, Dick Karp, Jim Larus, Ed Lazowska, Tsu-Jae King Liu, Luc Moreau, David Parnas, Peggie Rimmer, Sue Sentance, Robert Szlizs, Sam Watson, and Pat Yongpradit for helpful comments and suggestions.

In spite of the assistance of all of the above, Tony Hey and Gyuri Pápay wish to acknowledge that it is only the authors who are responsible for any remaining errors or omissions in the manuscript.

We also thank David Jou of Cambridge University Press and Rebecca Reid and Diane Aboulafia of GreatWork Strategic Communications for their assistance with figure permissions. We thank Shari Chappell of Cambridge University Press and our copy editor Christine Dunn of Dunn Write Editorial for helping shape our manuscript into a form suitable for publication.

Acknowledgments

Finally, we both wish to thank our editor, Lauren Cowles at Cambridge University Press, for her belief in our project and for her extreme patience as deadlines flew by. She also went far beyond the call of duty and read and edited the entire manuscript. The book is undoubtedly a better book because of her diligence.

Prologue: Blog entry from Jonathan Hey

FRIDAY, 27 JANUARY 2012

Key moments in tech history

I can still, and perhaps will always, remember:

- The first day we connected our NES to our TV and Mario appeared
- The first day I instant messaged a friend using MSN Messenger from France to England
- The first day I was doing a presentation and said I could get online without a cable
- The first day I was carrying my laptop between rooms and an email popped up on my computer
- The first day I tentatively spoke into my computer and my friend's voice came back
- The first day the map on my phone automatically figured out where I was

Each of these moments separately blew my mind on the day. It was like magic when they happened. The closest I have had recently was probably the first successful call using Facetime and waving my hand at a Kinect sensor. (Another, that most people probably haven't experienced, was watching a glass door instantly turn opaque at the touch of a button. Unbelievable.)

Each of these moments blew me away because things happened that weren't even part of my expectations. I expect our expectations these days have now risen sufficiently high that it'll probably take teleportation to get a similar effect from a teenager. Maybe life would be more fun if we kept our expectations low?

Beginnings of a revolution

Computer science also differs from physics in that it is not actually a science. It does not study natural objects. Neither is it, as you might think, mathematics. Rather, computer science is about getting something to do something....

Richard Feynman¹

What is computer science?

It is commonplace to say that we are in the midst of a computing revolution. Computers are impacting almost every aspect of our lives. And this is just the beginning. The Internet and the Web have revolutionized our access to information and to other people. We see computers not only providing intelligence to the safety and performance of such things as cars and airplanes, but also leading the way in mobile communications, with present-day smart phones having more computing power than leading-edge computers only a decade ago. This book tells the story how this all came about, from the early days of computers in the mid-1900s, to the Internet and the Web as we know it today, and where we will likely be in the future.

The academic field of study that encompasses these topics draws from multiple disciplines such as mathematics and electronics and is usually known as *computer science*. As Nobel Prize recipient, physicist Richard Feynman says in the quotation that introduces this chapter, computer science is not a science in the sense of physics, which is all about the study of natural systems; rather, it is more akin to engineering, since it concerns the study of man-made systems and ultimately is about getting computers to do useful things. Three early computing pioneers, Allen Newell, Alan Perlis, and Herbert Simon, were happy to use science to describe what they did, but put forward a similar definition to Feynman: computer science is the study of computers. As we shall see, computer science has much to do with the management of complexity, because modern-day computers contain many billions of active components. How can such complex systems be designed and built? By relying on the principles of *hierarchical abstraction* and *universality*, the two main themes that underlie our discussion of computers.

Hierarchical abstraction is the idea that you can break down the design of a computer into layers so that you can focus on one level at a time without having to worry about what is happening at the lower levels of the hierarchy. Feynman in his *Lectures on Computation* makes an analogy with geology and the



Fig. 1.1 The famous geological map of Great Britain devised by William “Strata” Smith (1769–1839). Smith was a canal and mining engineer who had observed the systematic layering of rocks in the mines. In 1815, he published the “map that changed the world” – the first large-scale geological map of Britain. Smith was first to formulate the superposition principle by which rocks are successively laid down on older layers. It is a similar layer-by-layer approach in computer science that allows us to design complex systems with hundreds of millions of components.

work of William Smith, the founder of stratigraphy – the branch of geology that studies rock layers and layering (Fig. 1.1). While the layering approach used in computer science was not inspired by geological layers, Feynman’s analogy serves as a useful memory hook for explaining hierarchical layers of computer architecture by reminding us that we can examine and understand things at each level (Fig. 1.2). This is the key insight that makes computers comprehensible.

Universality is linked to the notion of a universal computer that was introduced by Alan Turing and others. Turing suggested a very simple model for a computer called a Universal Turing Machine. This uses instructions encoded on a paper tape divided into sections with a very simple set of rules that the machine is to follow as the instruction in each section is read. Such a machine would be horribly inefficient and slow at doing complex calculations; moreover, for any specific problem, one could design a much more efficient, special-purpose machine. Universality is the idea that, although these other computers may be faster, the Universal Turing Machine can do any calculation that they can do. This is known as the Church-Turing thesis and is one of the cornerstones of computer science. This truly remarkable conjecture implies that your laptop, although much, much slower than the fastest supercomputer, is in principle just as powerful – in the sense that the laptop can do any calculation that can be done by the supercomputer!

So how did we get to this powerful laptop? Although the idea of powerful computational machines dates to the early nineteenth century, the direct line to today’s electronic computers can be traced to events during World War II (1939–1945).

A chance encounter

There are many detailed histories of the origins of computing, and it would take us too far from our goal to discuss this history in detail. Instead, we will concentrate only on the main strands, beginning with a chance meeting at a train station.

In 1943, during World War II, the U.S. Army had a problem. Their Ballistic Research Laboratory (BRL) in Aberdeen, Maryland, was falling badly behind in its calculations of firing tables for all the new guns that were being produced. Each new type of gun needed a set of tables for the gunner that showed the correct angle of fire for a shell to hit the desired target. These trajectory calculations were then being carried out by a machine designed by MIT Professor Vannevar Bush. This was the differential analyzer (Fig. 1.3). It was an analog device, like the slide rules that engineers once used before they were made obsolete by digital calculators, but built on a massive scale. The machine had many rotating disks and cylinders driven by electric motors and linked together with metal rods, and had to be manually set up to solve any specific differential equation problem. This setup process could take as long as two days. The machine was used to calculate the basic trajectory of the shell before the calculation was handed over to an army of human “computers” who manually calculated the effects on this trajectory of other variables, such as the wind speed and direction. By the summer of 1944, calculating



Fig. 1.2 This sponge cake is a further analogy of abstraction layers. It is most certainly more appealing to our senses than the rock layers of geological periods.



Fig. 1.3 Vannevar Bush's Differential Analyzer was a complicated analog computer that used rotating discs and wheels for computing integrals. The complete machine occupied a room and linked several integration units connected by metal rods and gears. The Differential Analyzer was used to solve ordinary differential equations to calculate the trajectories of shells at the U.S. Army Ballistics Research Laboratory in Aberdeen, Maryland.



B.1.1 John Mauchly (1907–80) and Presper Eckert (1919–95) were the designers of ENIAC. With John von Neumann, they went on to propose the EDVAC, a design for a stored-program computer, but unfortunately their future efforts were complicated by legal wrangling over intellectual property and patents. As a result, they left the Moore School at the University of Pennsylvania and set up a company to build the UNIVAC, the first successful commercial computer in the United States.

these tables was taking far too long and the backlog was causing delays in gun development and production. The situation seemed hopeless since the number of requests for tables that BRL received each week was now more than twice its maximum output. And this was after BRL had doubled its capacity by arranging to use a second differential analyzer located in the Moore School of Electrical Engineering at the University of Pennsylvania in Philadelphia. Herman Goldstine was the young army lieutenant in charge of the computing substation at the Moore School. And this was why he happened to be on the platform in Aberdeen catching a train back to Philadelphia on an evening in August 1944.

It was in March 1943 that Goldstine had first heard of a possible solution to BRL's problems. He was talking to a mechanic at the Moore School and learned of a proposal by an assistant professor, John Mauchly (**B.1.1**), to build an electronic calculator capable of much faster speeds than the differential analyzer. Mauchly was a physicist and was originally interested in meteorology. After trying to develop a weather prediction model he soon realized that without some sort of automatic calculating machine this task was impossible. Mauchly therefore developed the idea of building a fast electronic computer using vacuum tubes.

Goldstine was a mathematician by training, not an engineer, and so was not aware of the generally accepted wisdom that building a large-scale computer with many thousands of vacuum tubes was considered impossible because of the tubes' intrinsic unreliability. After talking with Mauchly, Goldstine asked him to submit a full proposal for such a vacuum-tube machine to BRL for funding. Things moved fast. Mauchly, together with the smartest graduate of the school, J. Presper Eckert, gave a presentation on their new proposal in Aberdeen less than a month later. They got their money – initially \$150,000 – and Project PX started on June 1, 1943. The machine was called the ENIAC, usually taken to stand for the Electronic Numerical Integrator And Computer.

It was while he was waiting for his train back to Philadelphia that Goldstine caught sight of a man he recognized. This was the famous mathematician John von Neumann (**B.1.2**), whom Goldstine had heard lecture on several occasions in his research as a mathematician before the war. As he later wrote:

It was therefore with considerable temerity that I approached this world-famous figure, introduced myself and started talking. Fortunately for me von Neumann was a warm, friendly fellow who did his best to make people feel relaxed in his presence. The conversation soon turned to my work. When it became clear to von Neumann that I was concerned with the development of an electronic computer capable of 333 multiplications per second, the whole atmosphere of our conversation changed from one of relaxed good humor to one more like an oral examination for a doctor's degree in mathematics.²

Soon after that meeting, Goldstine went with von Neumann to the Moore School so that von Neumann could see the ENIAC (**Fig. 1.4**) and talk with Eckert and Mauchly. Goldstine remembers Eckert's reaction to the impending visit:

He [Eckert] said that he could tell whether von Neumann was really a genius by his first question. If this was about the logical structure of the machine,

The Computing Universe



Fig. 1.4 A section of the original ENIAC machine on display at the University of Pennsylvania.

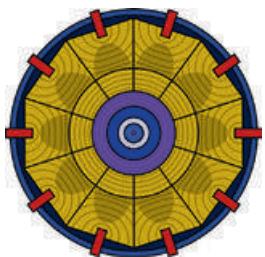


Fig. 1.5 A schematic diagram of the spherical implosion lens required to start the nuclear reaction in a plutonium bomb. John von Neumann's search for an automatic device that would speed up the complex calculations needed to model the lens led to his interest in ENIAC.

he would believe in von Neumann, otherwise not. Of course this was von Neumann's first query.³

The reason why von Neumann was so interested in the ENIAC was because of his work for the Manhattan atom bomb project at Los Alamos, New Mexico. The physicists at Los Alamos had a bottleneck in their schedule to produce a plutonium bomb. This was due to the complex calculations needed to model the spherical implosive lens for the bomb (Fig. 1.5). The lens was formed by accurately positioned explosives that produced a spherical compression wave. The wave would then compress the plutonium at the center of the sphere to criticality and thereby start the nuclear chain reaction. Von Neumann had asked Bush's Office of Scientific Research and Development (OSRD) for suggestions as to how this calculational bottleneck could be removed. He was advised to look at three automatic calculator projects that OSRD was funding that might deliver the increased computing power he needed. By the time he met Goldstine, von Neumann had concluded that none of the suggested projects, which included the Mark I, an electromechanical computer created by IBM and Howard Aiken at Harvard, would be of any help. The OSRD had made no mention of the Army-funded ENIAC project, since this was regarded by Bush and others as just a waste of money. The ENIAC team were therefore glad to welcome the famous von Neumann into their camp, and they had regular discussions over the next few months.

The ENIAC was completed in November 1945, too late to help the war effort. It was eight feet high, eighty feet long, and weighed thirty tons. It contained approximately 17,500 vacuum tubes, 70,000 resistors, 10,000 capacitors, 1,500 relays, and 6,000 manual switches. It consumed 174 kilowatts of power – enough to power several thousand laptops. Amazingly, only fifty years later, all of this monster amount of hardware could be implemented on a single chip (Fig. 1.6). Fortunately, the vacuum tubes turned out to be far more reliable than



B.1.2 John von Neumann (1903–57) was born in Budapest in the family of a wealthy banker. After graduating with a PhD in mathematics from Budapest ELTE and a diploma in chemical engineering from Zurich ETH, he won a scholarship in Gottingen and worked with David Hilbert on his ambitious program on the “axiomatization” of mathematics. In 1933, von Neumann was offered an academic position at the Institute for Advanced Study in Princeton, and was one of the institute's first four professors.

Von Neumann's extraordinary talent for mathematics and languages was evident from early in his childhood. At university, his teacher George Polya at the ETH in Zurich said of him:

He is the only student of mine I was ever intimidated by. He was so quick. There was a seminar for advanced students in Zurich that I was teaching and von Neumann was in the class. I came to a certain theorem, and I said it is not proved and it may be difficult. Von Neumann did not say anything but after five minutes he raised his hand. When I called on him he went to the blackboard and proceeded to write down the proof. After that I was afraid of von Neumann.³¹

Von Neumann was a genuine polymath who made pioneering contributions to game theory, quantum mechanics, and computing. He also hosted legendary cocktail parties, but his driving skills apparently left something to be desired:

Von Neumann was an aggressive and apparently reckless driver. He supposedly totaled a car every year or so. An intersection in Princeton was nicknamed “Von Neumann Corner” for all the auto accidents he had there.³²

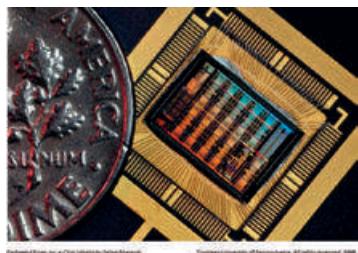


Fig. 1.6 The ENIAC on a chip. This chip was designed to mark the fiftieth anniversary of the ENIAC project by a group of students at the University of Pennsylvania. This 0.5 cm^2 chip can do the same computations as the original 30-ton computer in 1946. No other technology in the course of human history has achieved this pace of development.

anyone had expected. The calculational speed of the ENIAC was impressive – it was more than a thousand times faster than Aiken's Mark I machine. On ten-digit numbers, the machine could calculate more than five thousand additions or three hundred multiplications per second! However, although this was very much faster than the differential analyzer and the Mark I in terms of its basic operations, it still took about two days to set up the ENIAC to solve a specific problem – and this was after the operators had written a program specifying the correct sequence of operations.

Writing an ENIAC program required the programmer to have almost as much knowledge of the machine as its designers did (Fig. 1.7). The program was implemented by setting the ENIAC's switches to carry out the specific instructions and by plugging in cables to arrange for these instructions to be executed in the correct order. The six women who did most of the programming for the ENIAC were finally inducted into the Women in Technology International Hall of Fame in 1997 (Fig. 1.8).

The first problem to be performed by the ENIAC was suggested by von Neumann. The problem arose from his work at Los Alamos and involved the complex calculations necessary to evaluate a design for Edward Teller's proposed hydrogen bomb. The results revealed serious flaws in the design. Norris Bradbury, Director of the Los Alamos Laboratory, wrote a letter to the Moore School saying, “The complexity of these problems is so great that it would have been impossible to arrive at any solution without the aid of ENIAC.”⁴

Von Neumann and the stored-program computer

After the ENIAC design was finalized and the machine was being built, Eckert and Mauchly had time to think about how they could design a better computer using new memory storage technologies. It had become clear to them that the ENIAC needed the ability to store programs. This would enable programmers to avoid the lengthy setup time. Eckert and Mauchly probably came up with this idea for a stored-program computer sometime in late 1943 or early 1944. Unfortunately for them, they never got around to explicitly writing down their ideas in a specific design document for their next-generation computer. There are only some hints of their thinking in their progress reports on the construction of the ENIAC, but there now seems little doubt that they deserve at least to share the credit for the idea of the stored-program computer. When von Neumann first arrived at the Moore School in September 1944, he was briefed by Eckert and Mauchly about their ideas for a new machine they called EDVAC – Electronic Discrete Variable Computer. According to Mauchly's account, they told von Neumann the following:

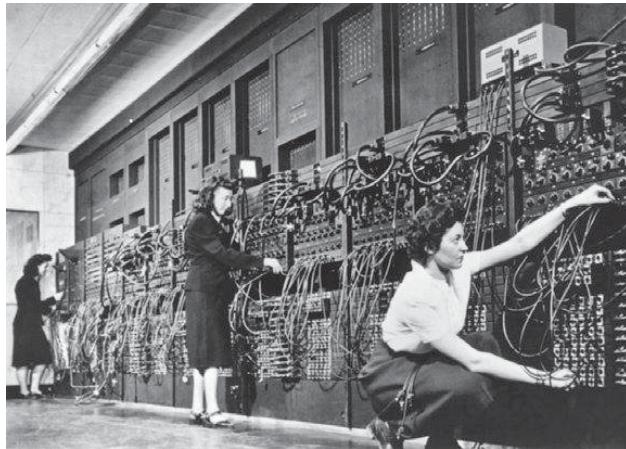
We started with our basic ideas: there would be only one storage device (with addressable locations) for the entire EDVAC, and this would hold both data and instructions. All necessary arithmetic operations would be performed in just one arithmetic unit (unlike the ENIAC). Of course, there would be devices to handle input and output, and these would be subject to the control module just as the other modules were.⁵

In the months that followed, the three of them refined their ideas for the EDVAC, which eventually resulted in von Neumann writing a paper, titled the



Fig. 1.7 U.S. Army ENIAC poster. The ENIAC was advertised as a work opportunity for mathematicians and puzzle solvers.

Fig. 1.8 The first programmers of ENIAC were women. In those days, programming meant setting all switches and rewiring the computer, a tedious operation that often took days to complete.



“First Draft of a Report on the EDVAC.” Although von Neumann had left blank spaces on his draft for the names of co-authors, unfortunately for Eckert and Mauchly, Goldstine went ahead and released the paper listing von Neumann as the sole author. The report contained the first description of the logical structure of a stored-program computer and this is now widely known as the von Neumann architecture ([Fig. 1.9](#)).

The first great abstraction in the report was to distinguish between the computer hardware and software. On the hardware side, instead of going into detail about the specific hardware technology used to build the machine, von Neumann described the overall structure of the computer in terms of the basic logical functions that it was required to perform. The actual hardware that performed these functions could be implemented in a variety of technologies – electromechanical switches, vacuum tubes, transistors, or (nowadays) modern silicon chips. All these different technologies could deliver the same computational capabilities, albeit with different performance. In this way, the problem of how the logical components are put together in a specific order to solve a particular problem has now been separated from concerns about the detailed hardware of the machine. This splitting of responsibilities for the hardware design and for the programming of the machine was the beginning of two entirely new engineering disciplines: computer architecture and software engineering.

For the hardware of the machine, von Neumann identified five functional units: the central arithmetic unit (CA), the central control unit (CC), the memory (M), the input (I), and the output (O) ([Fig. 1.10](#)). The CA unit carried out all the arithmetic and logical operations, and the CC unit organized the sequence of operations to be executed. The CC is the *conductor*, since it coordinates the operation of all components by fetching the instructions and data from the memory and providing clock and control signals. The CA’s task is to perform the required calculations. The memory was assumed to store both programs and data in a way that allowed access to either program instructions or data. The I/O units could read and write instructions or data into and out of the computer memory directly. Finally, unlike the ENIAC, which had used decimal arithmetic, von Neumann recommended that the EDVAC use binary arithmetic



Fig. 1.9 A Hungarian postage stamp that honors John von Neumann, complete with the mathematician’s likeness and a sketch of his computer architecture.

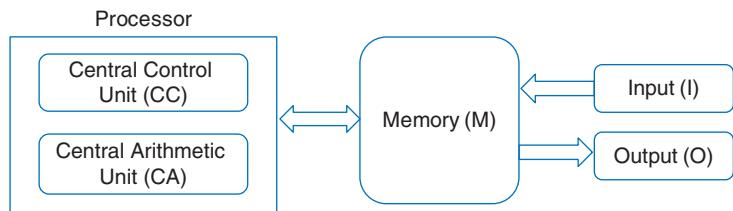


Fig. 1.10 The von Neumann Architecture. The main building blocks of all computers are the input, output, memory, and processor. The input (typically now a keyboard or a mouse) feeds data into the computer. This information is encoded by binary numbers and stored in the memory. The processor then fetches the information, decodes it, and performs the required calculations. The results are put back in the memory, where they can be read by the output device (typically a monitor, printer, or even a loudspeaker). The processor consists of two components: the Central Control Unit (CC) and the Central Arithmetic Unit (CA), now known as the Arithmetical and Logical Unit (ALU).

for its operations. As we shall see in [Chapter 2](#), binary, base-2 arithmetic is much better suited to efficient and simple electronic implementations of arithmetic and logic operations.

How does this von Neumann architecture relate to Turing's ideas about universality? Before the war, Turing had spent time in Princeton and von Neumann was well aware of the groundbreaking paper on theoretical computing machines he had completed as a student in Cambridge, UK. The memory, input, and output of von Neumann's abstract architecture are logically equivalent to the tape of a Universal Turing Machine, and the arithmetic and central control units are equivalent to the read/write component of Turing's logical machine. This means that no different computer design can do any different calculations than a machine built according to von Neumann's architecture. Instead of coming up with new architectures, computer engineers could spend their time optimizing and improving the performance of the von Neumann design. In fact, as we will see later, there are ways of improving on his design by eliminating the so-called von Neumann bottleneck – in which all instructions are read and executed serially, one after another – by using multiple processors and designing *parallel computers*.



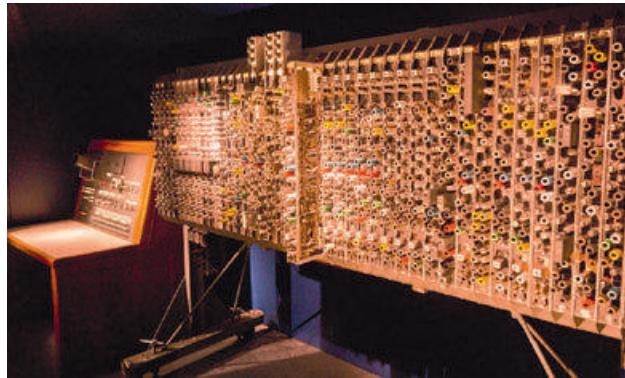
B.1.3 Leslie Comrie (1893–1950) was an astronomer and an expert on numerical calculations. He visited the Moore School in 1946 and brought the first copy of the EDVAC report back to Britain.

The global EDVAC diaspora

There were thirty-two people on the original mailing list for the “Report on the EDVAC” but news of the report soon spread far and wide. With World War II having come to an end, scientists were once again able to travel internationally, and by early 1946 the Moore School had already had several visitors from Britain. The first visitor from the United Kingdom to the Moore School was a New Zealander named Leslie Comrie ([B.1.3](#)). Comrie had a long-time interest in astronomy and scientific computation, and during the war he had led a team of scientists to computerize such things as bombing tables for the Allied Air Force. Remarkably, after his visit to see the ENIAC, Comrie was allowed to take a copy of the EDVAC report back to England. Back in England, he went to visit Maurice Wilkes (see [Timeline](#)) in Cambridge. Wilkes was a mathematical physicist who had returned from war service and was trying to

The Computing Universe

Fig. 1.11 The Pilot ACE was a computer with a distinctive flavor. Turing's design was much more detailed than that contained in von Neumann's EDVAC report published only three months earlier. Pilot ACE had many innovative features, such as three address instructions, variable-length block transfer, and bit-level manipulation, but it was difficult to program. This is one of the reasons why this unique design had little impact on the architecture of computers.



establish a viable computing laboratory in Cambridge. Wilkes recalls in his memoirs:

In the middle of May 1946 I had a visit from L.J. Comrie who was just back from a trip to the United States. He put in my hands a document written by J. von Neumann on behalf of the group at the Moore School and entitled "Draft Report on the EDVAC." Comrie, who was spending the night at St. John's College, obligingly let me keep it until the next morning. Now, I would have been able to take a Xerox copy, but there were then no office copiers in existence and so I sat up late into the night reading the report. In it, clearly laid out, were the principles on which the development of the modern digital computer was to be based: the stored program with the same store for numbers and instructions, the serial execution of instructions, and the use of binary switching circuits for computation and control. I recognized this at once as the real thing, and from that time on never had any doubt as to the way computer development would go.⁶

Another early visitor to the Moore School was J. R. Womersley from the U.K. National Physical Laboratory. Womersley had worked with differential analyzers and was duly impressed by the performance of the ENIAC. As a result of this visit, Womersley set about organizing a computing project at his laboratory and hired Turing to lead the team. Turing read von Neumann's report and then designed his own plan for a stored-program computer called ACE -Automatic Computing Engine (Figs. 1.11 and 1.12), where his use of the word *engine* was a deliberate homage to Charles Babbage. The ACE design report describes the concept for the machine in the following words:

£40,000 BRAIN IS A SCHOOLBOY'S DREAM WORKS OUT PROBLEMS LIKE LIGHTNING

"Evening News" Reporter

WHAT a brain! That is what we all said when we visited the National Physical Laboratory at Teddington to-day, to see Britain's latest computing "electronic brain" in action.

Now take a "simple" problem like 3,971,428,732 multiplied by 8,167,292,438. If you are in the skilled mathematician class it would take you eight minutes to work out that little problem. The brain which costs £40,000 takes one 500th of a second.

Schoolboys of the future will have an easy time if the scientist can only turn out a pocket-sized "brain."

Fig. 1.12 The London Evening News from November 28, 1950, reporting the speed of the Pilot ACE computer.

It is intended that the setting up of the machine for new problems shall be virtually only a matter of paper work. Besides the paper work nothing will have to be done except to prepare a pack of Hollerith cards in accordance with this paper work, and to pass them through a card reader connected to the machine. There will positively be no internal alterations to be made even if we wish suddenly to switch from calculating the energy levels of the neon atom to the enumeration of groups of order 720. It may appear puzzling that



Fig. 1.13 The Moore School of Electrical Engineering at the University of Pennsylvania, where the ENIAC was born.

this can be done. How can one expect a machine to do all this multitudinous variety of things? The answer is that we should consider the machine as doing something quite simple, namely carrying out orders given to it in a standard form which it is able to understand.⁷

This is not the last computing project to underestimate the difficulties associated with the “paper work” or, as we would now say, “programming the machine”!

In 1946, at the instigation of the new dean of the Moore School, Howard Pender, the Army Ordnance Department, and the U.S. Office of Naval Research sponsored a summer school on stored-program computing at the Moore School (Fig. 1.13). There were thirty to forty invitation-only participants mainly from American companies, universities, and government agencies. Alone among the wartime allies, Britain was invited to participate in the summer school. The Moore School Lectures on Computing took place over eight weeks in July and August, and besides Eckert and Mauchly, Aiken and von Neumann made guest appearances as lecturers. The first part of the course was mainly concerned with numerical mathematics and details of the ENIAC. It was only near the end of the course that security clearance was obtained that enabled the instructors to show the participants some details of the EDVAC design. Wilkes had received an invitation from Dean Pender and, despite funding and visa problems, decided it was worth going since he thought he was “not going to lose very much in consequence of having arrived late.”⁸ After attending the last two weeks of the school, Wilkes had time to visit Harvard and MIT before he left the United States. At Harvard he saw Howard Aiken’s Mark I and II electromechanical computers, and at MIT he saw a new version of Bush’s differential analyzer. He left the United States more convinced than ever that the future was not going to follow such “dinosaurs” but instead follow the route laid out by the EDVAC report for stored-program computers. On his return to Cambridge in England, Wilkes started a project to build the Electronic Delay Storage Automatic Calculator – usually shortened to EDSAC, in conscious homage to its EDVAC heritage.

The EDSAC computer became operational in 1949. In these early days of computing, a major problem was the development of suitable memory devices to store the binary data. Eckert had had the idea of using tubes filled up with mercury to store sound waves traveling back and forth to represent the bits of data, and Wilkes was able to successfully build such mercury delay line memory for the EDSAC. A variant on Wilkes’s design for the EDSAC was developed into a commercial computer called Lyons Electronic Office, or LEO. It was successfully used for running business calculations for the network of Lyon’s Corner Houses and Tea Shops. Wilkes later introduced the idea of microprogramming, which enabled complicated operations to be implemented in software rather than hardware. This idea significantly reduced the hardware complexity and became one of the key principles of computer design.

Meanwhile, back in the United States, Eckert and Mauchly had resigned from the Moore School after an argument over patent rights with the university and were struggling to get funding to build a commercial computer.



Fig. 1.14 Tom Kilburn and Freddie Williams with the “Baby” computer in Manchester. The machine had only seven instructions and had 32×32 bits of main memory implemented using a cathode ray tube.

After many difficulties, they ultimately succeeded in designing and building the famous UNIVAC (UNIVersal Automatic Computer) machine. With the war ended, von Neumann returned to Princeton and wasted no time getting funds to build an EDVAC architecture computer for the Institute for Advanced Study (IAS). He quickly recruited Goldstine and Arthur Burks from the EDVAC team and a talented engineer, Julian Bigelow, to help him design the IAS machine (see Timeline). In 1947, with Goldstine, von Neumann wrote the first textbook on software engineering called *Planning and Coding Problems for an Electronic Computing Instrument*.

While commercial interest in computers was beginning to develop in the United States, it was actually two teams in the United Kingdom that first demonstrated the viability of the stored-program computer. At Manchester, Freddie Williams and Tom Kilburn had followed the path outlined by von Neumann and in June 1948 they had a prototype machine they called Baby (see Timeline and Fig. 1.14). This ran the first stored program on an electronic computer on 21 June 1948. This success was followed in May 1949 by Wilkes’s EDSAC machine in Cambridge – which was undoubtedly the first stored-program computer with any significant computational power.

Key concepts

- Computation can be automated
- Layers and abstractions
- The stored program principle
- Separation of storage and processing
- Von Neumann architecture



Cartoon illustrating the requirement for calculating shell trajectories.

Some early history of computing

An idea long in the making

While the origins of the modern electronic computer can be traced back to EDVAC in the 1940s, the idea of powerful computational machines goes back much further, to the early nineteenth century and an Englishman named Charles Babbage.

Charles Babbage and the Difference Engine

The first government-funded computer project to overrun its budget was Charles Babbage's attempt to construct his Difference Engine in 1823. The project had its origins in the problem of errors in the mathematical tables of the *navigator's bible*, the British Nautical Almanac. These errors, either from mistakes in calculation or from copying and typesetting, plagued all such tables and were popularly supposed to be the cause of numerous shipwrecks. One study of a random selection of forty volumes of tables found three thousand errors listed on correction or errata sheets. Some sheets were even correction sheets for earlier corrections!

Charles Babbage (B.1.4) was a mathematician and a student at Cambridge University in 1812 when he first had the idea of using a machine to calculate mathematical tables. He wrote about the moment in his autobiography:

One evening I was sitting in the rooms of the Analytical Society, at Cambridge, my head leaning forward on the table in a kind of dreamy mood, with a table of logarithms lying open before me. Another member, coming into the room, and seeing me half asleep called out "Well, Babbage, what are you dreaming about?" to which I replied, "I am thinking that all these tables (pointing to the logarithms) might be calculated by machinery."¹⁰

Some years later Babbage was checking astronomical tables with his astronomer friend John Herschel. They each had a pile of papers in front of them containing the results for the tables as calculated by "computers." In those days, computers were not machines but people who had been given a precise arithmetical procedure to do the routine calculations by hand. The two piles contained the same set of calculations, each done by different computers but both should be identical. By comparing the results line by line Babbage and Herschel found a number of errors and the whole process was so slow and tedious that Babbage finally exclaimed "I wish to God these calculations had been executed by steam."¹⁰

As a result of his experience, Babbage spent the next few years designing what he called the Difference Engine – a mechanical machine that was able to calculate astronomical and navigational tables using a mathematical process called the method of constant differences. Correct calculations were only part of the problem however, since the copying and typesetting of the results were equally error prone. In order to eliminate these errors, Babbage designed his machine to record the results on metal plates that could be used directly for printing. By 1822 he had built a small working prototype and made a proposal to the Royal Society that a large,



B.1.4 Charles Babbage (1792–1871) was the son of a wealthy banker. He studied mathematics at Cambridge and was the leader of a radical group of students that overthrew the negative legacy of Isaac Newton's approach to calculus on mathematics in England by introducing new notation and mathematical techniques from France and Germany. Babbage is now known for his pioneering work in computing, but he was also a prolific inventor. Among other things, he invented the ophthalmoscope, a cowcatcher, the seismograph, and a submarine propelled by compressed air. However, Babbage's computing engines were never completed, and he died a disappointed man.

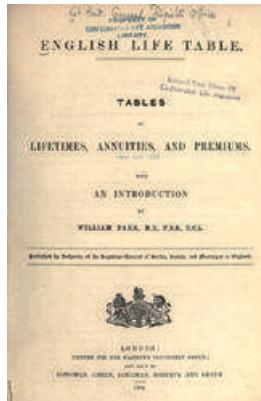


Fig. 1.15 The front page of *English Life Table* from 1864, portions of which were composed on a machine inspired by Babbage's Difference Engine. Unfortunately the machine lacked some of Babbage's error-protection mechanisms and this severely hampered its usefulness.

full-scale Difference Engine be built. As a result of his proposal, Babbage was encouraged to seek funding from the U.K. government, and he eventually received what was then an unprecedented offer of £1,500 in financial support. To give some idea of how this amount compares in today's prices, Doron Swade in his book on the Difference Engine states that "A gentleman in 1814, for example, could expect to support his wife and a few children in modest comfort on an income of £300 a year."¹¹ So in 1822, this represented a very significant investment by the government.

The full-scale engine would need thousands of precisely engineered gears and axles. Babbage therefore had to begin the project by spending a great deal of time with his skilled engineering draughtsman, Joseph Clement, devising and designing better machine tools capable of producing hundreds of identical parts. It was unfortunate for Babbage that he was building his machine at a time when manufacturing technology in Britain was in transition between individual craft manufacture and mass-production methods. A symptom of this was that there were no standards for even simple components like screws. One of Clement's engineers, Joseph Whitworth, later played a major role in the adoption of the standard Whitworth screw thread by the U.K. manufacturing industry.

By 1832, almost ten years and £10,000 after the project began, Babbage and Clement had built a small part of the machine, complete except for the printing mechanism. It worked perfectly and was able to deliver calculations to six-digit accuracy. Alas, at this point the project was brought to a standstill by a dispute between Babbage and Clement and work on the machine was never resumed. By the time of the formal cancellation of the project in 1842, the total cost to the U.K. government was more than £17,000; by comparison, a new steam engine built by Robert Stephenson for shipping to the United States in 1831 had cost less than £800.

The first article giving any substantial details of Babbage's design for his Difference Engine appeared in 1834, written by a colorful character called Dionysius Lardner and published under the title "Babbage's Calculating Engine" in the *Edinburgh Review*. After reading this lengthy and somewhat eulogistic article, George Scheutz, a Stockholm printer, publisher, and journalist, took up the cause of mechanized calculations. Without having access to full details of Babbage's mechanisms for the various operations, Scheutz invented his own. What is more, by the summer of 1843, he and his son Edvard had actually built a working machine, including the printing mechanism. Although they were originally concerned about Babbage's reaction to their efforts, the truth is that Babbage was delighted and set about helping them market their machine. In 1857, after winning a gold medal at the 1855 Great Exposition in Paris, one machine was sold for £1,000 to Dr. Benjamin Gould, the director of the Dudley Observatory in Albany, New York, as part of a flagship project to make the Albany observatory the "American Greenwich." Gould used the machine to calculate a set of tables for the orbit of the planet Mars. Alas, in 1859 Gould was fired and Scheutz's Tabulating Machine eventually was donated to the Smithsonian Museum.

Another machine was purchased by the United Kingdom's General Register Office. William Farr, the chief statistician, wanted to use the machine to automate the production of the tables of life expectancy, annuities, insurance premiums, and interest payments that he needed for the 1864 *English Life Table* (Fig. 1.15). The machine was built by Donkin & Company in London and was used for the English Life Table project. However, because the machine did not have all of Babbage's careful error-protection mechanisms, it proved to need constant care and attention. At the end of the project, only twenty-eight of the six hundred pages of printed tables in the *English Life Table* were entirely composed by the machine and 216 were partially



Fig. 1.16 Postage stamp issued to mark the bicentenary of Babbage's birth.

composed. The conclusion was that the machine had failed to deliver any significant benefits or cost savings. A sad postscript to this story is that both Scheutz and his son ended their lives bankrupt, a condition at least partly caused by their overenthusiasm for their calculating engines.

There is a positive postscript to this story. Babbage's son, Henry, inherited most of the unused parts for the Difference Engine that had been manufactured by Clements. Although many of the parts went for scrap, Henry saved enough parts to assemble six small demonstration machines. He sent these to several universities including Cambridge, University College London, and Manchester in the United Kingdom and Harvard in the United States. In the late 1930s, Howard Aiken, who with IBM pioneered the development of the Harvard Mark I, one of the early electromechanical computers, discovered the small demonstration engine sent to Harvard. He later said that he "felt that Babbage was addressing him personally from the past."¹²

It was not until 1991 that Doron Swade and a team at the Science Museum in the United Kingdom unveiled a working model of a full-scale Difference Engine built following Babbage's designs (Fig. 1.16). This demonstration showed conclusively that such machines could have been built in Babbage's day, albeit with a huge amount of engineering effort (see Fig. 1.17).

The Analytical Engine

Babbage was not a master of tact. Instead of finishing his Difference Engine, he unwisely suggested that the government abandon work on the still incomplete original machine and build a much more powerful and versatile machine he called the Analytical Engine. The Difference Engine was essentially a special-purpose calculator, and Babbage had realized that he could design a much more general-purpose machine capable of performing any arithmetical or logic operation. He conceived the idea for this vastly more powerful and flexible machine between 1834 and 1836 and kept tinkering and improving his design until the day he died. This Analytical Engine was never built and was therefore only a thought experiment. Nevertheless, its design



Fig. 1.17 In 1991, to mark the bicentenary of Babbage's birth, Doron Swade and his colleagues at the London Science Museum unveiled the Difference Engine II, a working model constructed according to Babbage's original designs. The computing historians and engineers went to great lengths to preserve the authenticity, using the original drawings, materials, and precision of manufacturing available in Babbage's time. They considered this work as the continuation of Babbage's project, but almost 150 years later. Difference Engine II contains about eight thousand cogs and weighs about 4.5 tons. It is operated by a crank handle, as can be seen in the accompanying photo of Swade cranking the machine. The design included numerous failsafe features, such as mechanical parity checking that prevented errors occurring even when some cogs get deranged due to vibrations. The cogwheels were manufactured so that they could fracture in a controlled way; this is an equivalent of a mechanical fuse. It must be said, however, that the machine was overdimensioned. It can calculate to an accuracy of forty-four binary digits. This looks excessive, especially if we consider that many of the machines we use today calculate accurately only to thirty-two binary digits. The machine can also calculate up to seventh-order polynomials, today we usually use third-order polynomials. It is not clear why Babbage thought he needed this level of accuracy; a simpler machine would have saved him many cogs and certainly would have made the construction much easier. Babbage saw the computer as an integral part of the calculation process, which he envisaged as a factory that produces numbers.



Fig. 1.18 A photograph of Jacquard's Loom showing the punched cards encoding the instructions for producing intricate patterns. The program is a sequence of cards with holes in carefully specified positions. The order of the cards and the positions of these holes determine when the needles should be lifted or lowered to produce the desired pattern.

captured many of the principles to be found in today's computers. In particular, Babbage's design separated the section of the machine where the various arithmetical operations were performed from the area where all the numbers were kept before and after processing. Babbage named these two areas of his engine using terms borrowed from the textile industry: the *mill* for the calculations, which would now be called the central processing unit or CPU, and the *store* for storing the data, which would now be called computer memory. This separation of concerns was a fundamental feature of von Neumann's famous report that first laid out the principles of modern computer organization.

Another key innovation of Babbage's design was that the instructions for the machine - or program as we would now call them - were to be supplied on punched cards. Babbage got the idea of using punched cards to instruct the computer from an automatic loom (Fig. 1.18) invented in France by Joseph-Marie Jacquard (B.1.5). The cards were strung together to form a kind of tape and then read as they moved through a mechanical device that could sense the pattern of holes on the cards. These looms could produce amazingly complex images and patterns. At his famous evening dinner parties in London, Babbage used to show off a very intricate silk portrait of Jacquard that had been produced by a program of about ten thousand cards.

Babbage produced more than six thousand pages of notes on his design for the Analytical Engine as well as several hundred engineering drawings and charts indicating precisely how the machine was to operate. However, he did not publish any scientific papers on the machine and the public only learned about his new ambitious vision through a presentation that Babbage gave in 1840 to Italian scientists in Turin. The report of the meeting was written up by a remarkable young engineer called Luigi Menabrea - who later went on to become a general in the Italian army and then prime minister of Italy.



B.1.5 Joseph-Marie Jacquard (1752–1834) (left) and Philippe de la Salle (1723–1804) pictured on a mural in Lyon (Mur des Lyonnais). Philippe de la Salle was a celebrated designer, who made his name in the silk industry. Jacquard's use of punched cards to provide the instructions for his automated loom inspired Babbage, who proposed using punched cards to program his Analytical Engine.

Ada Lovelace

It is at this point in the story that we meet Augusta Ada, Countess of Lovelace (B.1.6), the only legitimate daughter of the Romantic poet, Lord Byron. Lovelace first met Babbage at one of his popular evening entertainments in 1833 when she was seventeen. Less than two weeks later, she and her mother were given a personal demonstration of his small prototype version of his computing engine. Unusually for women of the time, at the insistence of her father, Ada had had some mathematical training. After this first meeting with Babbage, Ada got married and had children but in 1839 she wrote to Babbage asking him to recommend a mathematics tutor for her. Babbage recommended

Augustus De Morgan, a well-known mathematician who had made significant contributions to both algebra and logic. Ada had a very high opinion of her own abilities and wrote to Babbage that “the more I study, the more insatiable do I feel my genius for it to be.”¹³ Her opinion of her talent is supported, in part at least, by a letter written by De Morgan to her mother. In the letter, De Morgan suggested that Ada’s abilities could lead her to become “an original mathematical investigator, perhaps of first-rate eminence.”¹⁴

At the suggestion of a mutual friend, the scientist Charles Wheatstone, Lovelace translated Menabrea’s paper for publication in English. Babbage then suggested that she add some notes of her own to the paper. He took a great interest in her work and gave her the material and examples he had used in his talk in Turin and helped her by annotating drafts of her notes. Babbage also wrote a completely new example for her: the calculation of the Bernoulli numbers (a complex sequence of rational numbers) using the Analytical Engine.

Although Ada did not originate this example, she clearly understood the procedure well enough to point out a mistake in Babbage’s calculation. She both amplified Babbage’s ideas and expressed them in her own forthright manner, as is evident from these two examples from her notes:

The distinctive characteristic of the Analytical Engine, and that which has rendered it possible to endow mechanism with such extensive faculties as bid fair to make this engine the executive right-hand of abstract algebra, is the introduction into it of the principle which Jacquard devised for regulating, by means of punched cards, the most complicated patterns in the fabrication of brocaded stuffs. It is in this that the distinction between the two engines lies. Nothing of the sort exists in the Difference Engine. We may say most aptly that the Analytical Engine weaves algebraical patterns just as the Jacquard-loom weaves flowers and leaves....

Many persons ... imagine that because the business of the Engine is to give its results in numerical notation the nature of its processes must consequently be arithmetical and numerical, rather than algebraical and analytical. This is an error. The engine can arrange and combine its numerical quantities exactly as if they were letters or any other general symbols; and in fact it might bring out its results in algebraic notation, were provisions made accordingly.¹⁵

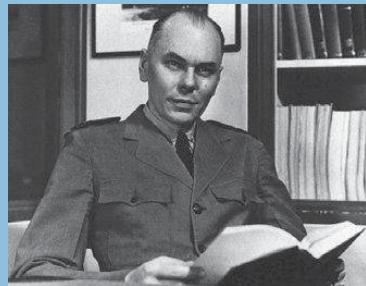
Babbage certainly had not published or developed the idea of using his machine for algebra in any detail. One remark from Lovelace is also often quoted in debates about artificial intelligence and computers: “The Analytical Engine has no pretensions whatever to originate anything. It can do whatever we know how to order it to perform.”¹⁶ We will look later at the question of whether computers are capable of recognizable intelligence.



B.1.6 A portrait of Ada Lovelace (1815–52) drawn by a robotic artist. Her father, the Romantic poet Lord Byron, was instrumental in ensuring that she was educated in mathematics. She was the first to write in English about the potential capabilities of Babbage’s Analytical Engine and is considered by some to be the first “computer programmer.” She was certainly the first to emphasize that the machine could manipulate symbols as well as perform numerical calculations. She also wrote that perhaps one day machines would even be able to write poetry.



TL.1.1. John Vincent Atanasoff (1903–95) with Clifford Berry; they constructed ABC using vacuum tubes.



TL.1.2. Howard Aiken (1900–73), the constant clicking of relays created a sound as if the “room was full of knitting ladies.”



TL.1.3. The first stored program computer. The memory was constructed from Cathode Ray Tubes.

ABC
1936

ZI
1934

MARK-I
1944

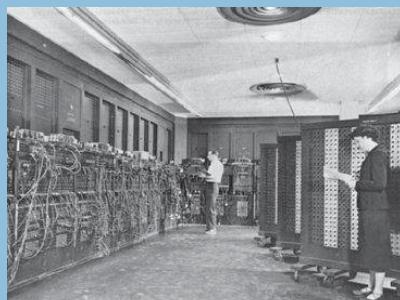
ENIAC
1945

Manchester Baby
1948

EDSAC
1949



TL.1.4 Konrad Zuse tinkering with his Z-computer.



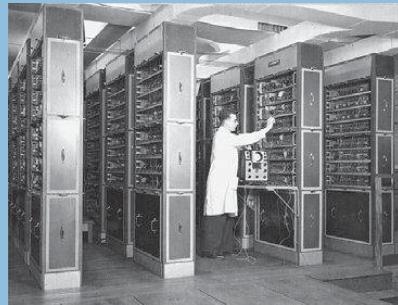
TL.1.5 Rewiring the ENIAC was a challenging task.



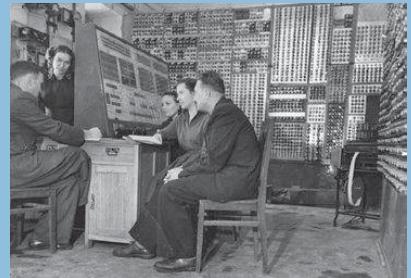
TL.1.6 Maurice Wilkes checking the valves of the EDSAC computer.



TL.1.7 The first stored-program computer constructed in Australia used a mercury line memory and could generate music.



TL.1.8 LEO, a successful business computer, used by a chain of Lyon's tea shops.



TL.1.9 The first Soviet computer was built in a monastic hostel near Kiev.

CSIR Mark I
1949

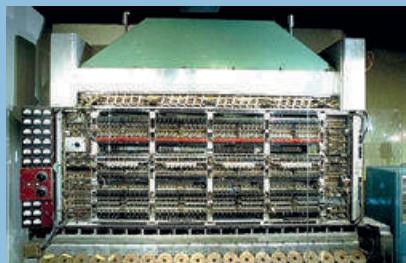
IAS
1952

LEO
1951

Whirlwind
1951

MESM
1951

UNIVAC
1952



TL.1.10 The IAS machine was a prototype for many computers because the design was not patented and was widely disseminated. The programs running on the computer were mainly calculations for the H-bomb, but also biological simulations.



TL.1.11 The first flight simulator computer was used for training bomber crews.



TL.1.12 Presper Eckert (center) demonstrating the UNIVAC to CBS reporter Walter Cronkite. The machine was used to predict the results of the 1952 U.S. election, but even the programmers did not believe their (correct) prediction, made after only 7% of the vote was in: a landslide win for Eisenhower instead of the close election predicted by the pollsters.

Code breakers and bread makers



Fig. 1.19 Memorial to Polish code breakers at Bletchley Park. Their contribution was critical to the development of the Bombe machines used to break the Enigma codes.



Fig. 1.20 A photograph of the Colossus computer, the code-breaking machine that nobody knew existed until many years after the war. It was designed and built by Tommy Flowers, an engineer at the British Post Office in 1943.

machines – that showed it was possible to make a device to break the Lorenz codes. This was followed by the ULTRA project to build an all-electronic version of Heath Robinson called Colossus (Fig. 1.20). Although this machine was certainly not a general-purpose computer, it had 1,500 vacuum tubes as well as tape readers with optical sensors capable of processing five thousand teleprinter characters a second. The machine was designed and built by Tommy Flowers (B.1.8), an engineer at the U.K. Post Office's Dollis Hill research laboratory in London, and became operational in December 1943, more than two years before the ENIAC. One of the great achievements of Colossus was reassuring the Allied generals, Eisenhower and Montgomery, that Hitler had believed the deception that the D-Day invasion fleet would come from Dover. The immense contribution of code breakers was recognized by Winston Churchill when he talked about “the Geese that laid the golden eggs but never cackled.”¹⁷

The main task for the code breakers was to read the text from a paper tape and to work out the possible settings of the twelve rotors of the encrypting device. Colossus was first demonstrated in December 1943

No history of the early days of computing would be complete without recounting the pioneering work of the British cryptologists at Bletchley Park and the development of the first computer dedicated to business use.

Bletchley Park, Enigma, and Colossus

During World War II, British mathematicians and scientists had been looking to automated machines for their top-secret work on code breaking at Bletchley Park. Both Turing and his Cambridge mentor Max Newman (B.1.7) were intimately involved in designing and building automated machines to assist in decrypting secret German military communications. Turing was involved in deciphering messages encrypted using the famous Enigma machine. With a head start given to them by Polish Intelligence (Fig. 1.19), Turing helped develop electromechanical machines, known as *bombe*s, which were used to determine the settings of the Enigma machine. These machines were operational as early as 1940 and contributed greatly to protecting convoys from U-boats in the North Atlantic.

The German High Command in Berlin used an even more complex cipher machine called Lorenz. Newman's team built a machine – called Heath Robinson after a popular cartoonist who drew eccentric



B.1.7 Max Newman (1897–1984) was a brilliant Cambridge, U.K., mathematician and code breaker. It was Newman's lectures in Cambridge that inspired Alan Turing to invent his famous Turing Machine. Newman was at Bletchley Park during World War II and his team was working on the messages encrypted by the Lorenz cipher machine. They built a machine – called Heath Robinson – to break the Lorenz code, and this was later automated as the Colossus computer.



B.1.8. The name of Tommy Flowers (1905–98) is virtually unknown to most students of computing. His immense contribution to computing and code breaking during the war has only recently emerged from the obsessive secrecy imposed on the code-breaking activities at Bletchley Park after World War II. Flowers built an electronic code-breaking machine called Colossus, which was capable of breaking the so-called Lorenz cipher used by the German high command. Instead of using electromechanical devices as in the bombes used for breaking the Enigma codes, Flowers decided to use vacuum tubes. This idea initially met with some resistance because it was generally thought that tubes would not be sufficiently reliable. Colossus contained about one and a half thousand vacuum tubes and was the world's first special-purpose electronic computer. Flowers described the heat generated by the computer with the following words: "Ah, the warmth at two a.m. on a damp, cold English winter!"¹³

and provided invaluable information for preparing the Normandy landing. The automated code-breaking devices such as Colossus and the bombes made a significant contribution to shortening the war (Fig. 1.21).

At the end of the war, Winston Churchill gave orders that most of the ten Colossus machines should be destroyed. Flowers personally burned the blueprints in a furnace at the Dollis Hill laboratory. The reason for this destruction was so that the British government could promote and sell Enigma-like machines to other governments yet still retain the ability to decipher messages sent using the machines! Two Colossus machines were in use at the UK Government Communication Headquarters (GCHQ) in Cheltenham until the late 1950s. With the coming of digital communications, the need for such secrecy about the wartime activities at Bletchley Park became unnecessary, and information about Colossus began to emerge in the 1970s. A secret 1945 report on the decrypting of the Lorenz signals was declassified in 2000 and contains the following description of working with Colossus:

It is regretted that it is not possible to give an adequate idea of the fascination of a Colossus at work; its sheer bulk and apparent complexity; the fantastic speed of thin paper tape round the glittering pulleys; ... the wizardry of purely mechanical decoding letter by letter (one novice thought she was being hoaxed); the uncanny action of the typewriter in printing the correct scores without and beyond human aid....”¹⁴

One clear result of this U.K. obsession for secrecy about its achievements in computer development during the war years was that all subsequent computer developments, even in the United Kingdom, were based on von Neumann's EDVAC design for stored-program computers.



Fig. 1.21. “We Also Served” – a memorial to code breakers at Bletchley Park. On the back of the memorial is a quote from Winston Churchill, written in Morse code: “My Most Secret Source.”

LEO: The first business computer

A curious footnote to the EDSAC is the development of Lyons Electronic Office, or LEO, the world's first computer specifically designed for business applications rather than for numerical calculations. The unlikely business was that of J. Lyons & Co., which ran a nationwide chain of Lyons Tea Shops as well as Lyons Corner Houses (Fig. 1.22). These offered English high teas and cream cakes in London and featured uniformed waitresses called "Nippies." The catering business required an army of clerks to ensure that the correct quantity of baked goods was delivered fresh every day and to process the associated receipts and invoices. It seems obvious to us now that such jobs can be computerized, but at the time it required real vision to recognize that a computer originally designed to calculate trajectories of shells could be useful for nonscientific business applications.



B.1.9 John Pinkerton (1919–97), one of the first computer engineers, pictured in front of the LEO, the first business computer that he designed and built as a modified version of the EDSAC. The machine was built for J. Lyons & Co. to automate the record keeping for the production and delivery of their baked goods to their famous Lyons Tea Shops. LEO, Lyons Electronic Office, went into operation in 1951.

November. The machine calculated the value of the bakery output in terms of bread, cakes, and pies from the bakery input of materials, labor, and power costs. It used the factory costs with the prices and profit margins to calculate the value of the products distributed to the teashops, grocers, and restaurants. LEO also calculated the value of products kept in stock. The LEO Computers Company was formed in 1954 and delivered upgrades of the machine until the early 1960s, when the company merged with English Electric. In 1968 this company formed the foundation for a new British computer company called International Computers Ltd., which operated profitably for several decades.



Fig. 1.22 Lyons operated a network of tea shops and "corner houses" throughout the United Kingdom and, surprisingly, pioneered the use of computers for business calculations.

Other Beginnings

Computer development was not limited to the United States and the United Kingdom; other countries, including Germany, the Soviet Union, and Australia, also pioneered the development of digital electronic computers.

Konrad Zuse, the Z series, and Plankalkül

In Germany, Konrad Zuse (B.1.10) is now widely acknowledged as one of the founding fathers of computing. He worked on his designs in isolation during the difficult times leading up to the war years. In 1941, his first operational electromechanical computer, the Z3, contained some architectural features that, independently of Zuse's work, have become key principles for computer designers. Unlike the ENIAC, Zuse's machine used binary encoding for both data and instructions, significantly reducing the complexity of the design. The Z3 was the first automatic, program-controlled, relay computer, predating Howard Aiken's Mark I machine. By 1943, Zuse was constructing a new Z4 computer. At the height of the wartime bombing of Germany, Zuse managed to transport the Z4 from Berlin and hide it in a stable in a small village in the Bavarian Alps. Zuse also developed a programming language called Plankalkül (plan calculus). In this language he introduced the concept of assignment and loops that are now seen as key components of programming languages. After the war, in 1949, Zuse founded a computer company, Zuse KG; he sold his first Z4 to ETH in Zurich in 1950. His company continued to develop new machines; he sold 56 of his Z22 vacuum tube machines for industrial and university research. The company was bought by Siemens in 1967.

Sergei Lebedev, MESM, and BESM

Sergei Alekseyevich Lebedev (B.1.11) was one of the pioneers of Soviet computing. Under his leadership after the war, a secret electronic laboratory was established in the outskirts of Kiev, where he and his team started to build the first Soviet computers. By December 1951, they had a functioning machine, and this marked the beginning of indigenous Soviet computers. They produced computers that ranged from large mainframe computers of the classes BESM, URAL, and Elbrus to smaller machines such as MIR and MESM. These names are largely unknown outside of Russia, but in scientific and engineering circles behind the Iron Curtain they were held in great respect. The BESM computers formed the backbone of Soviet computing; about 350 were produced. BESM-1 was built in 1953, and the last of this series, BESM-6, in 1966. However, in 1967 a political decision was taken to copy IBM machines. This was the end of indigenous Soviet computing and a bitter disappointment for many of the Soviet computer pioneers.



B.1.10 Konrad Zuse (1910–95) independently designed and constructed computers during World War II. Until recently, his pioneering work was not widely known although IBM had taken an option on his patents in 1946. His programming language, Plankalkül, was never implemented in Zuse's lifetime. A team from the Free University of Berlin produced a compiler for the language in 2001.



B.1.11 Sergei Alekseyevich Lebedev (1902–74) was the founder of the Soviet computer industry. A keen alpinist as well as a brilliant engineer, he climbed Europe's highest peak, Mount Elbrus. In 1996 he was posthumously awarded the Charles Babbage medal by the IEEE society. His name in Cyrillic script is written Сергей Алексеевич Лебедев.

Few specifics are known about the Soviet machines, since most of the documents were never published. There is a general perception that computing in the Soviet Union in the 1950s and 1960s was far less developed than in the West. However, it is hard to imagine that the Soviets would have been able to achieve the spectacular results in space exploration, defense, and technology without possessing some serious computing capacity. Doron Swade, a senior curator of the London Science Museum, traveled to Siberia in 1992 to procure a Soviet BESM-6 computer for his museum's collection. In an interview for BBC Radio 4, he was asked about Lebedev's contribution and the MESM computer:

Was MESM original? I would say almost completely yes. Was its performance comparable? Certainly. Was BESM's performance comparable? I'd say BESM by that stage was being outperformed by the equivalent generation in the [United] States. But as a workhorse, as an influential machine in the plenty of Russian computer science in terms of its utility and its usefulness to the space program, to military research and scientific research it is probably, arguably the most influential machine in the history of modern computing.¹⁹

Trevor Pearcey and the CSIR Mark I

Trevor Pearcey (B.1.12) was born in the United Kingdom and had worked on applying advanced mathematics to radar development during World War II. In 1945, he emigrated to Australia and visited Harvard and MIT on the way; he saw both Aiken's Mark I and Bush's Differential Analyzer in operation. By 1946 Pearcey was working at the Division of Radiophysics of the Australian Council for Scientific and Industrial Research (CSIR) located at the University of Sydney. He understood the limitations of the machines he had seen in the United States and saw the potential for using vacuum tubes to create a high-speed digital computer. By the end of 1947, Pearcey, working on the theory, and Maston Beard, an electrical engineering graduate from Sydney working on the hardware, had defined their design. Although Pearcey visited the United Kingdom near the end of 1948 and saw the Manchester Baby and the Cambridge EDSAC, he saw no reason to change his original design. He later asserted that the CSIR Mark I "was completely 'home-grown' some 10,000 miles distant from the mainstream development in the UK and USA."²⁰ As with all the early computers, the development of computer memory technology was one of the major challenges. It was left to engineer Reg Ryan on the Australian team to design the memory system for the CSIR Mark I using mercury delay lines. The machine operated at 1 kilohertz and its delay line memory could store 768 words, each 20 bits long. By the end of 1949, their computer was able to run some basic mathematical operations and could genuinely claim to be one of the first operational stored-program computers. By 1951, CSIR had changed its name to the Commonwealth Scientific and Industrial Research Organisation, and the computer became the CSIRO Mark I. At Australia's first conference on Automatic Computing Machines in August 1951, the Mark I gave the first demonstration of computer-generated music by playing the popular wartime song "Colonel Bogey." In 1954, the CSIRO project was officially ended, and the machine was transferred to the University of Melbourne in 1955. The university's new Computation Laboratory was opened in 1956 with the CSIRO machine as its workhorse, rechristened CSIRAC. The machine ran for the next eight years, with only about 10 percent of its running time taken up for maintenance.



B.1.12 Trevor Pearcey (1919–98) was born in London and graduated from Imperial College with a degree in physics and mathematics. After working on radar systems during the war, he emigrated to Australia and was responsible for designing and building the CSIR Mark I at the University of Sydney. This was one of the world's first computers to use vacuum tubes.

2 The hardware

I always loved that word, Boolean.

Claude Shannon¹

Going through the layers

In the last chapter, we saw that it was possible to logically separate the design of the actual computer hardware – the electromagnetic relays, vacuum tubes, or transistors – from the software – the instructions that are executed by the hardware. Because of this key abstraction, we can either go down into the hardware layers and see how the basic arithmetic and logical operations are carried out by the hardware, or go up into the software levels and focus on how we tell the computer to perform complex tasks. Computer architect Danny Hillis says:

This hierarchical structure of abstraction is our most important tool in understanding complex systems because it lets us focus on a single aspect of a problem at a time.²

We will also see the importance of “functional abstraction”:

Naming the two signals in computer logic 0 and 1 is an example of functional abstraction. It lets us manipulate information without worrying about the details of its underlying representation. Once we figure out how to accomplish a given function, we can put the mechanism inside a “black box,” or a “building block” and stop thinking about it. The function embodied by the building block can be used over and over, without reference to the details of what’s inside.³

In this chapter, like Strata Smith going down the mines, we’ll travel downward through the hardware layers (Fig. 2.1) and see these principles in action.

George Boole and Claude Shannon

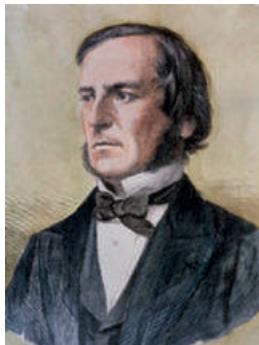
In the spring of 1936, Vannevar Bush was looking for a smart electrical engineering graduate who could assist visiting scientists in setting up their calculations on his Differential Analyzer at MIT. Claude Shannon (B.2.1), a

Processor	Memory	I/O
Registers and logic circuits		
Logic gates		
Electrons		

Fig. 2.1. A diagram showing the major abstraction layers of computer hardware.



B.2.1. Claude Shannon (1916–2001) is often referred to as the father of information technology. He is credited with two groundbreaking ideas: the application of Boolean algebra to logical circuit design and digitization of information. In this photograph, he is holding a mechanical mouse that can learn from experience as it moves around the complicated maze.



B.2.2. George Boole (1815–64) invented the algebra of 0s and 1s by introducing the rules and notation for describing logic. He was a self-taught mathematician and at the age of thirty-four became a professor of mathematics at Queens College in Cork, Ireland. Boole was widely recognized for his work aiming to combine algebra and logic. De Morgan, the leading logician of the time wrote: “Boole’s system of logic is but one of many proofs of genius and patience combined.”¹¹

twenty-year-old graduate from the University of Michigan, applied and got the position. Although the Differential Analyzer was a mechanical machine with many spinning discs and rods, there was, as Shannon said later, also “a complicated control circuit with relays.”⁴ A relay is just a mechanical switch that can be opened or closed by an electromagnet, so that it is always in one of two states: either on or off, just like a light switch. Bush suggested that a study of the logical organization of these relays could be a good topic for a master’s thesis. Shannon agreed, and drawing on his undergraduate studies in symbolic logic, he began trying to understand the best way to design complicated relay circuits with hundreds of relays. Commenting on the importance of symbolic logic in this endeavor, Shannon later said that “this branch of mathematics, now called Boolean algebra, was very closely related to what happens in a switching circuit.”⁵ Let’s see how this comes about.

George Boole ([B.2.2](#)) was a nineteenth-century, self-taught mathematician whose best-known work is a book called *An Investigation of the Laws of Thought*. In this book, Boole tried to reduce the logic of human thought to a series of mathematical operations in which decisions are predicated on determining whether any given logical statement is true or false. It was the Greek philosopher Aristotle who introduced what is now called propositional logic. This was a form of reasoning that enabled him to deduce new conclusions by combining true propositions in a “syllogism” of the form:

Every Greek is human.
Every human is mortal.
Therefore, every Greek is mortal.

Boole devised a language for describing and manipulating such logical statements and for determining whether more complex statements are true or false. Equations involving such statements can be written down using the logical operations AND, OR, and NOT. For example, we can write an equation to express the obvious fact that if neither statement A nor statement B is true, then both statements A and B must be false:

$$\text{NOT } (\text{A OR B}) = (\text{NOT A}) \text{ AND } (\text{NOT B})$$

This equation is actually known as De Morgan’s theorem, after Boole’s colleague, Augustus De Morgan ([B.2.3](#)). It is a simple, (but as we shall see) powerful expression of Boolean logic. In this way, Boolean algebra allows much more complex logical statements to be analyzed.

Shannon realized that relays combined in circuits were equivalent to combining assertions in Boolean algebra. For example, if we connect two relays, A and B, in series and apply a voltage to the ends, current can only flow if both relays are closed ([Fig. 2.2](#)). If we take the closed state of a relay as corresponding to “true,” then this simple connection of two relays corresponds to an AND operation: both relays must be closed – both true – for current to flow. Similarly, if we connect up the two relays in parallel, this circuit performs an OR operation, since current will flow if either relay A or B is closed or true ([Fig. 2.3](#)).

Shannon’s master’s thesis, “A Symbolic Analysis of Relay and Switching Circuits,” showed how to build electrical circuits that were equivalent to

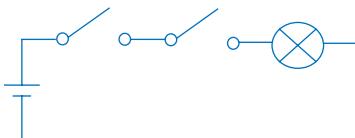


Fig. 2.2. Two relay switches in series serving as an AND gate. The current can only flow if both relays are closed.



B.2.3. Augustus De Morgan (1806–71) is known for his pioneering work in logic, including the formulation of the theorem that bears his name.

expressions in Boolean algebra. Relays are switches that can be closed or open corresponding to a state being logical true or false, respectively. This means that any function that can be described as a correct logical statement can be implemented as a system of electrical switches. Toward the end of his thesis, Shannon points out that true and false could equally well be denoted by 1 and 0, respectively, so that the operation of the set of switches corresponds to an operation in binary arithmetic. He wrote:

It is possible to perform complex mathematical operations by means of relay circuits. Numbers may be represented by the position of relays and stepping switches, and interconnections between sets of relays can be made to represent various mathematical operations.⁶

As an example, Shannon showed how to design a relay circuit that could add two binary numbers. He also noted that a relay circuit can make comparisons and take alternative courses of actions depending on the result of the comparison. This was an important step forward, since desktop calculators that could add and subtract had been around for many years. Shannon's relay circuits could not only add and subtract, they could also make decisions.

Let us now look at how Shannon's insights about relay circuits and Boolean algebra transformed the way early computer builders went about designing their machines. Computer historian Stan Augarten says:

Shannon's thesis not only helped transform circuit design from an art into a science, but its underlying message – that information can be treated like any other quantity and be subjected to the manipulation of a machine – had a profound effect on the first generation of computer pioneers.⁷

But before we take a deeper look at relay circuits and Boolean algebra, we should say a few words about binary arithmetic.

Binary arithmetic, bits, and bytes

Although some of the early computers – like the ENIAC – used the familiar decimal system for their numerical operations, it turns out to be much simpler to design computers using binary arithmetic. Binary operations are simple, with no need to memorize any multiplication tables. However, we pay a price for these easy binary operations by having to cope with longer numbers: a dozen is expressed as 12 in decimal notation but as 1100 in binary (B.2.4, Fig. 2.4).

Mathematics in our normal decimal system works on base 10. A number is written out in “positional notation,” where each position to the left represents 10 to an increasing power. Thus when we write the number 4321 we understand this to mean:

$$4321 = (4 \times 10^3) + (3 \times 10^2) + (2 \times 10^1) + (1 \times 10^0)$$

Here 10^0 is just 1, 10^1 is 10, 10^2 is 100, and so on; the numbers of powers of ten in each field are specified by digits running from 0 to 9. In the binary system, we use base 2 instead of base 10, and we specify numbers in powers of two and use only the two digits, 0 and 1. Thus the binary number 1101 means:

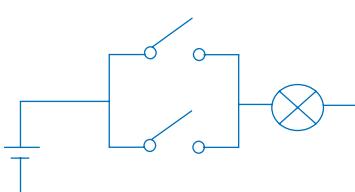


Fig. 2.3. Two relay switches in parallel equivalent to an OR gate. The current can flow if at least one of the relays is closed.



Gottfried Wilhelm Leibniz
1646 - 1716

B.2.4. Gottfried Wilhelm Leibniz (1646–1716) was a German mathematician and philosopher who is credited with the discovery of binary numbers. Leibniz described his ideas in a book titled *The Dyadic System of Numbers*, published in 1679. His motivation was to develop a system of notation and rules for describing philosophical reasoning. Leibniz was fascinated by binary numbers and believed that the sequences of 0s and 1s revealed the mystery of creation of the universe and that everything should be expressed by binary numbers. In 1697 he wrote a letter to the Duke of Brunswick suggesting a design for a celebration of binary numbers to be minted in a silver coin. The Latin text at the top of the coin reads “One is sufficient to produce out of nothing everything.” On the left of the table there is an example of binary addition and on the right, an example of multiplication. At the bottom the text in Latin says the “Image of Creation.”

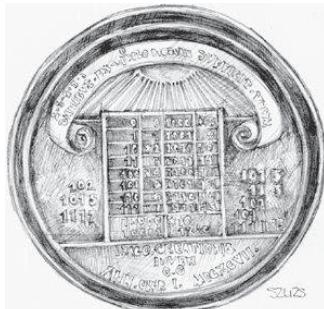


Fig. 2.4. A silver coin capturing the concept of binary numbers.

$$1101 = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 13 \text{ in decimal}$$

We can add and multiply binary numbers in a similar but much simpler way as in the decimal system. When adding in the decimal system, we automatically align the powers of ten in the numbers and perform a “carry” to the next power when needed, for example:

$$\begin{array}{r} 47 \\ + 85 \\ \hline 132 \end{array}$$

For binary addition we have similar sum and carry operations. Adding the decimal numbers 13 and 22 using binary notation leads to:

$$\begin{array}{r} 1101 \\ + 10110 \\ \hline 100011 \end{array}$$

We have just used the basic rules of binary addition:

$$\begin{aligned} 0 + 0 &= 0 \\ 0 + 1 &= 1 \\ 1 + 0 &= 1 \\ 1 + 1 &= 0 \text{ plus a carry 1} \end{aligned}$$

Let's take a look at multiplication in both the decimal and binary systems by multiplying 37 by 5 in both decimal and binary notation:

In decimal:

$$\begin{array}{r} 37 \\ \times 5 \\ \hline 185 \end{array}$$

In binary:

$$\begin{array}{r}
 100101 \\
 \times 101 \\
 \hline
 100101 \\
 10010100 \quad (\text{shifted two places to the left}) \\
 \hline
 10111001
 \end{array}$$

As this example shows, all we need to perform binary multiplication is an implementation of these basic rules of binary addition including the carry and shift operations.

We can perform a binary addition physically by using a set of strips of plastic with compartments, rather like ice cube trays, and small pebbles to specify the binary numbers. An empty compartment corresponds to the binary digit 0; a compartment containing a pebble represents the binary digit 1. We can lay out the two strips with the numbers to be added plus a strip underneath them to hold the answer (Fig. 2.5).

The abstract mathematical problem of adding 1101 to 10110 has now been turned into a set of real world rules for moving the pebbles. Remarkably, with these simple rules we can now add two numbers of any size (Fig. 2.6). What this shows is that the basic operations of addition and multiplication can be reduced to a very simple set of rules that are easy to implement in a variety of technologies – from pebbles to relays to silicon chips. This is an example of functional abstraction.

The word *bit* was used by Shannon as a contraction of the phrase “binary digit” in his groundbreaking paper “A Mathematical Theory of Communication.” In this paper, Shannon laid the foundations for the new field of information theory. Modern computers rarely work with single bits but with a larger grouping of eight bits, called a “byte,” or with groupings of multiple bytes called “words.” The ability to represent all types of information by numbers is one of the groundbreaking discoveries of the twentieth century. This discovery forms the foundation of our digital universe. Even the messages that we have sent out into space are encoded by numbers (Fig. 2.7).

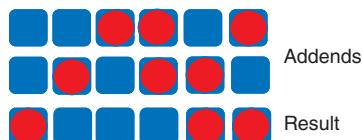


Fig. 2.5. Addition of binary numbers as illustrated by pebbles.



Fig. 2.6. A Russian abacus, used in almost all shops across the Soviet Union well into the 1990s. It is fast, reliable, and requires no electricity or batteries. It was common practice in shops that the results calculated by an electronic till were double-checked on the abacus.

Universal building blocks

With this introduction, we can now explain how a small set of basic logic building blocks can be combined to produce any logical operation. The logic blocks for AND and OR are usually called logic gates. These gates can be regarded just as “black boxes,” which take two inputs and give one output depending entirely on the inputs, without regard to the technology used to implement the logic gate. Using 1s and 0s as inputs to these gates, their operation can be summarized in the form of “truth tables.” The truth table for the AND gate is shown in Figure 2.8, together with its standard pictorial symbol. This table reflects the fundamental property of an AND gate, namely, that the output of A AND B is 1 only if both input A is 1 and input B is 1. Any other combination of inputs gives 0 for the output. Similarly, the truth table for the OR

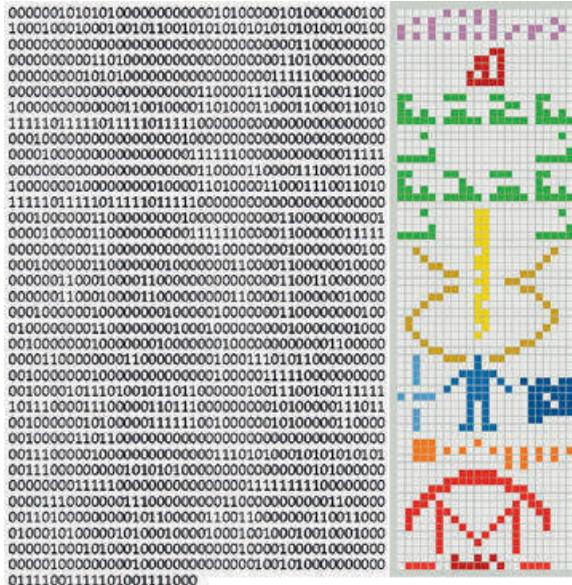


Fig. 2.7. This photograph shows the binary message that was sent out to space by the Arecibo radio telescope in November 1974. Using numbers to represent information is expected to be a universally understandable way to communicate with alien civilizations. The message shown here contains 1,679 binary digits arranged in seventy-three rows and twenty-three columns. The first ten rows represent numbers from 1 to 10 in binary format. The following rows are numbers that represent our genetic basis: the atomic numbers of chemical elements that form DNA, followed by the chemical formulas of the bases in DNA, the shape of the double helix, and the number of nucleotides in DNA. These are followed by the figure and height of an average man, the population of Earth, the position of the planet in our solar system, and the dimensions of the radio antenna broadcasting this message.

gates can be written as in [Figure 2.9](#), where we also show the usual pictorial symbol for an OR gate. For the output of the OR gate to be 1, either or both of the inputs A and B have to be 1.

For a complete set of logic gates from which we can build any logical operation, it turns out that we need to supplement these AND and OR gates by another, very simple gate, the NOT, or invert, operation (see [Fig. 2.10](#)). This is a black box that inverts the input signal. If the input is 1, the NOT gate outputs a 0; if the input is a 0, NOT outputs a 1. The truth table for the NOT gate is shown in with the usual symbol for NOT. De Morgan's theorem allows one to play around with these operators and find complicated equivalences between combinations of operators, such as writing an OR gate in terms of AND and NOT gates. Another example is joining the OR and NOT gates to construct a NOR gate (see [Fig. 2.11](#)).

Functional abstraction allows us to implement these logic building blocks in a range of technologies – from electromagnetic relays to vacuum tubes to transistors. Of course, different types of gates may be easier or harder to make in any given technology. The set of AND, OR, and NOT gates are sufficient to

Fig. 2.8. Truth table for AND Gate.

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1



Fig. 2.9. Truth table for OR Gate.

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1



Fig. 2.10. The NOT Gate.

A	NOT A
0	1
1	0

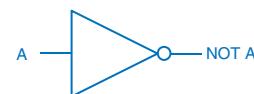


Fig. 2.11. The NOR Gate.

A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0



construct any logical operation with any number of inputs and outputs. There are also other possible choices for such a “complete set” of gates.

Let’s see how this all works by building a “bit-wise adder” black box from our basic AND, OR, and NOT building blocks. The bit-wise adder has two inputs – the bits A and B to be added – and two outputs – the result R and the carry C, if there is one (Fig. 2.12).

We need to combine AND, OR, and NOT gates appropriately to produce this logical behavior. First we note that the table for the carry bit is identical to that of the AND gate. We can write:

$$\text{Carry } C: A \text{ AND } B$$

The table for the result R is almost the same as that for the OR gate except for the result for the 1 + 1 input when the output must be inverted. The result R is therefore A OR B unless A AND B is 1. After some thought we can write this as:

$$\text{Result } R: (A \text{ OR } B) \text{ AND NOT } (A \text{ AND } B)$$

This leads to the implementation of the bit-wise adder shown in Figure 2.13.

This circuit is actually called a “half-adder” because although it correctly produces the carry value, it does not include a possible carry as an input to the device, as well as A and B. Using the half-adder of Figure 2.13, we can easily create an implementation for a “full adder” (Fig. 2.14). Adders for words with any number of bits can now be created by chaining these full adders together.

These examples illustrate two key principles of computer design. The first principle is that we rely on a “hierarchical design” process by which complex objects are built up from simpler objects. Logic gates are our fundamental and

The Computing Universe

Fig. 2.12. Truth table for bit-wise adder.

A	B	R	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Fig. 2.13. A half-adder made by combining four logic gates.

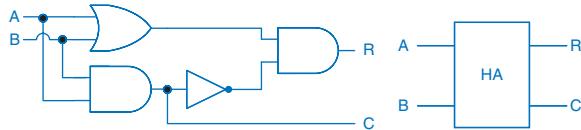


Fig. 2.14. A full adder made by composing two half-adders with an OR gate.

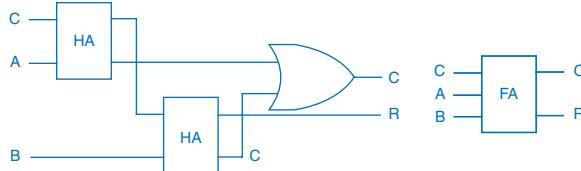


Fig. 2.15. The first binary adder, consisting of two battery cells, wires, two telephone relays, two light bulbs, pieces of wire, and a switch made from a tobacco tin.

universal building blocks. These objects can then be combined to build a bit-wise half-adder, which can then be used to build a full bit-wise adder, which then can be used to build whole-word adders and so on. The second principle is that of “functional abstraction.” We have seen how the early computers used electromagnetic relays to implement logic gates and logical operations (B.2.5, Fig. 2.15).

The slow relays were soon replaced by the faster but unreliable vacuum tubes, which in turn gave way to transistors and now to integrated circuits in silicon. The important point is that the logical design of an adder is independent of its implementation. In his delightful book *The Pattern in the Stone*, the computer architect Danny Hillis shows how you can make mechanical implementations of logic gates (Fig. 2.16).

The memory hierarchy

In order to be able to do something useful, computers need a mechanism for storing numbers. A useful memory not only needs to store results of intermediate calculations on some sort of digital scratch pad, but also needs to be



B.2.5. George Stibitz (1904–95). As is often the case in science, some of Claude Shannon’s ideas about relay circuits had been discovered independently around the same time. George Stibitz, a physicist at Bell Telephone Laboratories, was a member of a group of mathematicians whose job was to design relay-switching equipment for telephone exchanges. Stibitz also saw “the similarity between circuit paths through relays and the binary notation for numbers.”^{B2} Over one weekend in 1937, Stibitz wired up some relays to give the binary digits for the sum of two one-digit binary numbers. His output was two lightbulbs, which lit up according to the result of the binary addition. He then designed more complicated circuits that could subtract, multiply, and divide. With a Bell Labs engineer named Samuel Williams, Stibitz went on to build a machine with about four hundred relays that could handle arithmetic on complex numbers.

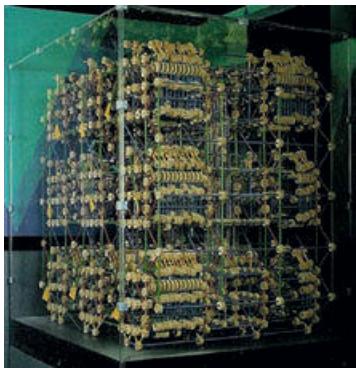


Fig. 2.16. We think of computers in terms of today's technologies, which use electronics and integrated circuits. Computers can be constructed from mechanical devices. This machine was built by a group of students at MIT using Tinker Toy spools and fishing line and can play tic-tac-toe.



Fig. 2.17. Pigeonholes are a useful analogy for a computer's memory.

able to alter what we have stored. We can think of computer memory as being like an array of mailboxes or pigeonholes (Fig. 2.17). Each box can store a data value that can either be retrieved or replaced with a new value. Memory registers are just a set of electronic boxes that can store a pattern of bits. As with logic gates, a wide variety of technologies have been used to implement computer memory. From an abstract point of view, the specific technology does not matter – in practice it matters a great deal, for reasons of reliability, speed of access, and cost!

In the logic-gate circuits discussed in the preceding text, the output states are completely determined by the inputs and the connections between the gates. Such a circuit is called a “combinational circuit.” It is also possible to construct another type of circuit – a “sequential circuit” – for which the output of a device depends on the previous history of its inputs. A counter circuit is an example of a sequential device where the current count number stored is the sum of the number of pulses it has received. The elemental sequential digital circuit is designed to be stable in one of two states. These “bistable elements” are usually called “flip-flops” – since they flip between the two states in response to an input. The basic flip-flop circuit is important because it is used as a memory cell to store the state of a bit. Register memories are constructed by connecting a series of flip-flops in a row and are typically used for the intermediate storage needed during arithmetic operations. Another type of sequential circuit is an oscillator or clock that changes state at regular time intervals. Clocks are needed to synchronize the change of state of flip-flop circuits.

The simplest bistable circuit is the set-reset or RS flip-flop (Fig. 2.18). The state of the flip-flop is marked as Q and is interpreted as the state 1 if the voltage at Q is high or as 0 if the voltage is low. The complement of Q , \bar{Q} , is also available as a second output. There are also two terminals that allow the flip-flop to be initialized. The state Q can be set to 1 by applying a voltage pulse on the “set” input S . A signal on the “reset” input R resets Q to 0. Figure 2.18 shows an RS flip-flop made out of NOR gates together with the corresponding truth tables. An input signal $S = 1$ sets $\bar{Q} = 0$ and if the input $R = 0$ then both inputs to the top NOR gate are zero. Thus a signal on the set line S and no signal on R gives $Q = 1$. This makes both inputs to the lower NOR gate 1. The other elements of the truth table can be filled in by similar reasoning. Note that an input state with both $R = 1$ and $S = 1$ is logically inconsistent and must be avoided in the operation of the flip-flop. So far, this RS flip-flop is still a combinational circuit because the state Q depends only on the inputs to R and S . We can make this a sequential RS flip-flop by adding a clock signal and a couple of additional gates (Fig. 2.19). The response of the clocked RS flip-flop at time $t+1$, $Q(t+1)$, now depends on the inputs and the state of the flip-flop at time t , $Q(t)$. A clock pulse must be present for the flip-flop to respond to its input states. There are many other different types of flip-flops and it is these bistable devices that are connected together to make registers, counters, and other sequential logic circuits.

Nowadays computers make use of a whole hierarchy of memory storage, built from a variety of technologies (Fig. 2.20). The earliest machines had only registers for storing intermediate results, but it soon became apparent that computers needed an additional quantity of memory that was less intimately linked to the central processing unit (CPU). This additional memory is called

Fig. 2.18. Truth table and schema of a flip-flop circuit built from NOR gates. Inputs $S = 1$ and $R = 1$ are not allowed, the outputs marked by * are indeterminate.

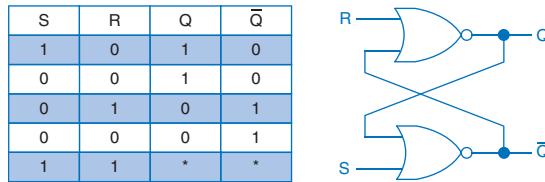
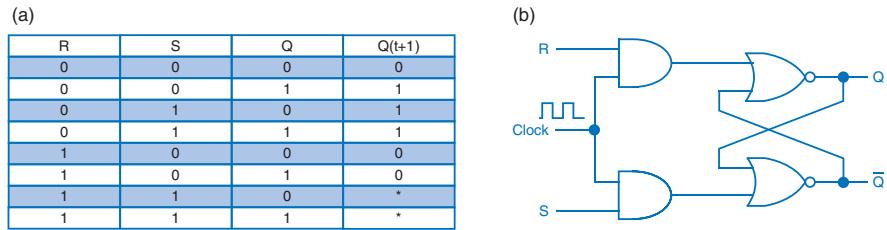


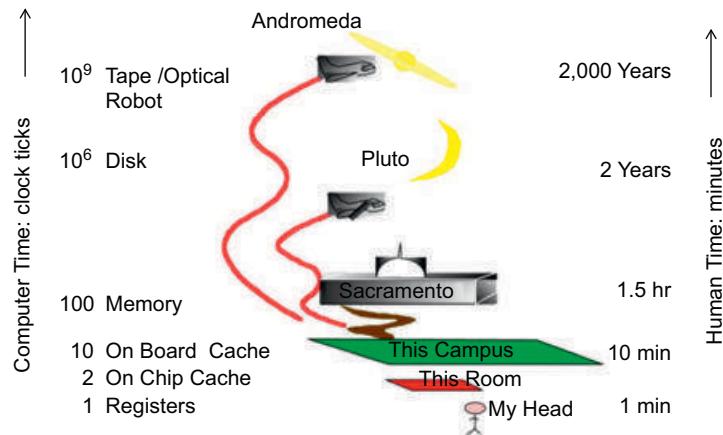
Fig. 2.19. Truth table and schema of clocked RS flip-flop.



the “main memory” of the computer and can be used to store results from registers as well as storing data needed by the registers for the different stages of the calculation. Between the registers and main memory, modern computers now incorporate several levels of memory that can be accessed more quickly than the main memory. These fast-access levels constitute the “cache memory” that is used to store the most frequently used data in order to avoid time delays that would be incurred in retrieving data from the slower memory.

Finally, since main memory is expensive, computer engineers introduced “secondary memory.” This uses cheaper and slower technologies but allows data to be transferred to the main memory as and when required. Initially, data for this secondary memory was recorded on punched cards or paper tape and fed in manually to the machine by computer operators. The use of cards for secondary memory was superseded by magnetic tapes rather than much more expensive magnetic core memory. Magnetic tapes holding computer data became so common that many movies showing computers in operation showed images of spinning tape drives as an icon for a computer. Much clever engineering has been devoted to making tape drives extremely reliable and fast. However, there is one problem that no amount of engineering can solve:

Fig. 2.20. How long does it take to get the data? This figure shows an analogy suggested by Jim Gray to illustrate the different data access times and the importance of memory hierarchy in computers. On the left, the access time is given in CPU clock ticks. For a typical 1 gigahertz clock this is one nanosecond. To relate these times to human timescales, on the right we translate a clock tick to one minute. The drawing in the middle illustrates how far we could have traveled during the time to retrieve data from the different elements of the computer memory hierarchy.



magnetic tapes are fundamentally sequential in the way they store and access data on the tape. Accessing some data stored halfway along the tape requires running through all the other data on the tape until that point is reached. This is fine if we only want to read long streams of information, but it is not very efficient if we want to access small amounts of nonsequentially stored bits of information. For this reason, disks and solid state semiconductor memory technologies that allow “random access” directly to any piece of data stored on the device are usually preferred, with the use of magnetic tapes restricted to providing archival backup storage for large data sets.

The fetch-execute cycle

We have now seen that computer hardware consists of many different components, which can all be implemented in a variety of ways. How do we coordinate and orchestrate the work of all these devices? The word *orchestrate* is an appropriate analogy. In an orchestra, there is a conductor who makes sure that everybody plays at the right time and in the right order. In the computer, an electronic clock performs the function of the conductor. Without the presence of a clock signal, memory circuits will not operate reliably. The clock signal also determines when the logic gates are allowed to switch.

In his draft report on the EDVAC, another key idea that von Neumann introduced was the “fetch-execute cycle.” This relies on an electronic clock to generate a heartbeat that drives the computer through its series of operations. For simplicity and reliability, von Neumann chose the simplest possible control cycle to be coordinated by the central control unit:

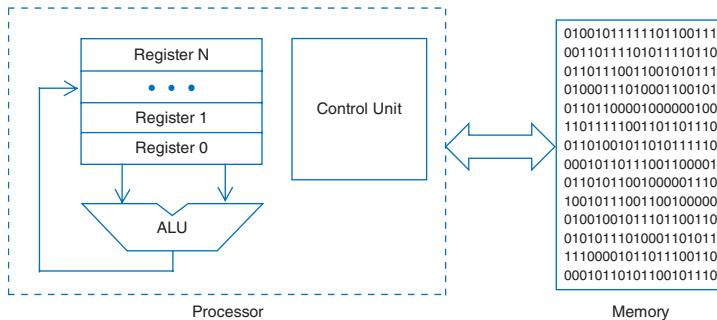
- Fetch the next instruction from memory and bring it to the control unit;
- Execute the instruction using data retrieved from memory or already present;
- Send results back for storage in memory; and
- Repeat the fetch-execute cycle.

Von Neumann chose the approach of just getting one instruction at a time because he feared that any other approach would make the computer too hard to build and program reliably. Alan Perlis, one of the early programming pioneers, once said, “sometimes I think the only universal in the computing field is the fetch-execute cycle.”⁸

The processor or CPU is the place where the instructions are executed and the data are manipulated. The main functions of the processor are to fetch the instructions from the main memory, decode the instructions, fetch the data on which the instruction’s mathematical or logical operation will be performed, execute the instructions, and store the results. These main functions have not changed since the early processor designs. On a logical level, a simple processor (see Fig. 2.21) consists of a bank of registers, an arithmetical logical unit (ALU), and control unit (CU).

The CU fetches instructions from the memory, decodes them and generates the sequence of control signals that are required for completing the instructions. The ALU performs the arithmetical and logical operations. For an execution of each instruction, various components need to be connected by switches

Fig. 2.21. A programmer's view of a processor.



that set the paths directing the flow of electrons. The bank of registers is for storing the instructions and the intermediate results of operations.

The exact choice of instructions that the hardware is built to execute defines the hardware-software interface. This is the “instruction set” of the machine. In the next chapter we will go up the hierarchy from the hardware layers and examine the software of the machine.

Key concepts

- Hierarchical design and functional abstraction
- Boolean algebra and switching circuits
- Binary arithmetic
- Bits, bytes, and words
- Logic gates and truth tables
- Combinational and sequential logic circuits
- Flip-flops and clocks
- The memory hierarchy
- The fetch-execute cycle



Some early history

The Manchester Baby and the Cambridge EDSAC computers

In his draft report on the design for the EDVAC, von Neumann analyzed the technical options for implementing the memory of the machine at length and concluded that the ability to construct large memories was likely to be a critical limiting factor. The first “stored program” computer to actually run a program was the University of Manchester’s “Baby” computer. This was a cut-down design of their more ambitious “Mark 1” and was built primarily to test the idea of hardware architect, Freddie Williams, to use a cathode ray tube – like the screens of early televisions – as a device for computer memory. In June 1948, the Baby ran a program written by co-architect Tom Kilburn to find the highest factor of 2^{18} (Fig. 2.22). The program was just a sequence of binary numbers that formed the instructions for the computer to execute. The output appeared on the cathode ray tube and, as Williams recounted, it took some time to make the system work:

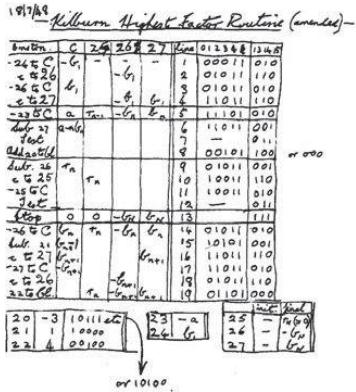


Fig. 2.22. Kilburn's highest factor routine from July 1948. The program ran for fifty-two minutes, and executed about 2.1 million instructions and made more than 3.5 million memory accesses.

When first built, a program was laboriously inserted and the start switch pressed. Immediately the spot on the display tube entered a mad dance. In early trials it was a dance of death leading to no useful result and, what was even worse, without yielding any clue as to what was wrong. But one day it stopped and there, shining brightly in the expected place, was the expected answer.⁹

The success of the Manchester Baby experiment led to the construction of the full-scale Manchester Mark 1 machine. This became the prototype for the Ferranti Mark I, the world's first “commercially available general-purpose computer”¹⁰ in February 1951, just a month before Eckert and Mauchly delivered their first UNIVAC computer in the United States.

While the Manchester Baby showed that a stored program computer was feasible, it was the EDSAC, built by Maurice Wilkes (B.2.6) and his team in Cambridge that was really the “first complete and fully operational regular electronic digital stored-program computer.”¹¹ The computer used mercury delay-lines for storage as reflected in Wilkes's choice of the name EDSAC – Electronic Delay-Storage Automatic Calculator. The development of a suitable computer memory technology was one of the major problems for the early computer designers. It was mainly because of storage difficulties that EDVAC-inspired computers in the United States were delayed and lagged behind the Manchester and Cambridge developments. Wilkes chose to use mercury delay-lines for EDSAC because he knew that such delay-lines had played an important role in the development of radar systems during the war. Wilkes had a working prototype by February 1947, just six months after he had attended the Moore School Lectures. Working with a very limited budget had forced Wilkes to make some compromises in the design: “There was to be no attempt to fully exploit the technology. Provided it would run and do programs that was enough.”¹²



B.2.6. Maurice Wilkes (1913–2010) seen here checking a mercury delay-line memory. He was a major figure in the history of British computing and at the University of Cambridge he led the team that designed and built the first fully operational stored-program computer.

Wilkes said later:

It resembled the EDVAC in that it was a serial machine using mercury tanks, but that was all. When I was at the Moore School the EDVAC design did not exist, except maybe in Eckert's head.¹³

His visit to the Moore School had had a huge impact on Wilkes and he left with an enduring respect for Eckert and Mauchly and said "They remain my idols."¹⁴ He also spent time with John Mauchly after the lectures and acknowledged this action as a "wonderful, wonderful piece of generosity."¹⁵

Computer memory technologies

Williams and Kilburn and their team designing the Manchester Baby developed an internal memory using cathode ray tubes, the same technology as was then used in radar screens and televisions. In these so-called Williams Tubes, the electron guns could make charge spots on the screen that corresponded to binary 1 and 0 values. Since the charge would dissipate after a fraction of a second, the screen needed to be refreshed in order to retain the record of the bits. The Baby had four Williams Tubes: one to provide storage for 32-by-32-bit words; a second to hold a 32-bit "register," in which the intermediate results of a calculation could be stored temporarily; and a third to hold the current program instruction and its address in memory. The fourth tube, without the storage electronics of the other three, was used as the output device and could display the bit pattern of any selected storage tube (Fig. 2.23). Williams Tubes were used as storage in the commercial version of the Baby, the Ferranti Mark 1, as well as in such U.S. computers as the Princeton IAS and the IBM 701.

Another early memory storage technology, originally suggested by Eckert, was based on the idea of a mercury delay-line – a thin tube filled with mercury that stores electronic pulses in much the same way as a hiker in a canyon can "store" an echo. A pulse represented binary 1; no pulse, binary 0. The pulses bounced from end to end and could be created, detected, and reenergized by electronic components attached to the tube. The EDSAC, built by Wilkes and his team in Cambridge, U.K., used mercury delay-lines. Wilkes had the good fortune to recruit a remarkable research physicist at Cambridge, Tommy Gold (B.2.7), who had been working on mercury delay-lines for radar during the war. Gold's knowledge and experience were invaluable in constructing large, five-foot "tanks" of mercury that were long enough to store enough pulses but were also precision engineered to an accuracy of a thousandth of an inch. Each mercury-filled tube could store 576 binary digits and the main store consisted of thirty-two such tubes, with additional tubes acting as central registers within the processor.

Mercury delay-lines and Williams Tubes were widely used until the early 1950s, when Jay Forrester (B.2.8), working at MIT, invented the magnetic core memory (Fig. 2.24). This device consisted of small

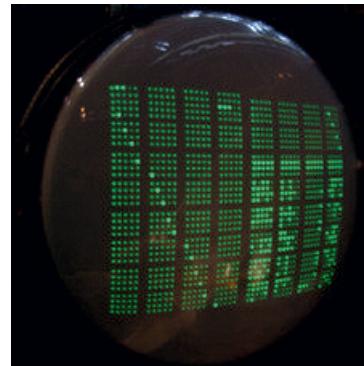


Fig. 2.23. A bit pattern on a cathode ray tube used in "Williams" memory.



B.2.7. Austrian-born astrophysicist Tommy Gold (1920–2004) did important work in many scientific and academic fields. His knowledge of mercury delay-line storage from his experience with radar during the war was very helpful to Wilkes. Gold made major contributions to astronomy and cosmology as well as to radar technology. Gold was first to propose the now generally accepted theory that pulsars are rotating neutron stars.

magnetizable rings located at the intersections of a grid of wires. Magnetization in the north direction could represent binary 1, and this could be flipped to a south magnetization, representing binary 0, by changing the current in the wire. By having a rectangular grid of wires and locating the cores at the intersections, it was possible to access each core individually. This allowed genuine random access of the individual memory locations – as opposed to having to go through the bits in sequence to get to the desired bit, as would be the case if the bits are stored on a magnetic tape. Forrester's technology was first tested in the construction of the Memory Test Computer at MIT in 1952. Compared to memory based on Williams Tubes, magnetic core memory proved much faster and far more reliable.

The first device to provide almost random access to data was not Forrester's magnetic core memory. It was a spinning drum with a magnetizable surface that allowed fast access to information stored in magnetized bands on the drum. This was invented in 1948 by Andrew Booth of Birkbeck College in England (B.2.9). Booth had made a visit to Princeton and had seen the progress von Neumann's team was making toward building the IAS stored program computer. Booth's prototype magnetic drum device was only two inches in diameter and could store ten bits per inch (Fig. 2.25). He followed up this prototype with larger drums that featured thirty-two magnetized bands, each divided into thirty-two words of thirty-two bits. A read/write head simply read off the values as the drum spun. Booth's drum memory was soon taken up by others and was adopted as secondary memory for Williams and Kilburn's scaled up version of the Baby, the Manchester Mark 1 machine. Magnetic drum memory was widely used for secondary memory in the 1950s and 1960s, until it was gradually superseded by magnetic disks.

The first hard disk was introduced by IBM in 1956 and hard disks soon became ubiquitous. Hard disk drives consist of a number of flat circular disks, called platters, mounted on a spindle (Fig. 2.26). The platters are coated with a thin film of magnetic material, and changes in the direction of magnetization record the required pattern of binary digits. The engineering of these drives is impressive. The platters can rotate at

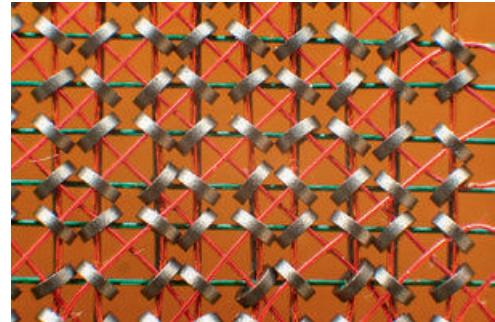


Fig. 2.24. Forrester's Magnetic Core Memory.



B.2.8. Jay Forrester holding a frame of core memory from the Whirlwind computer. He invented magnetic core memory while working at MIT in the early 1950s. His invention proved much faster and more reliable than the earlier Williams Tubes or delay-line memory technologies.

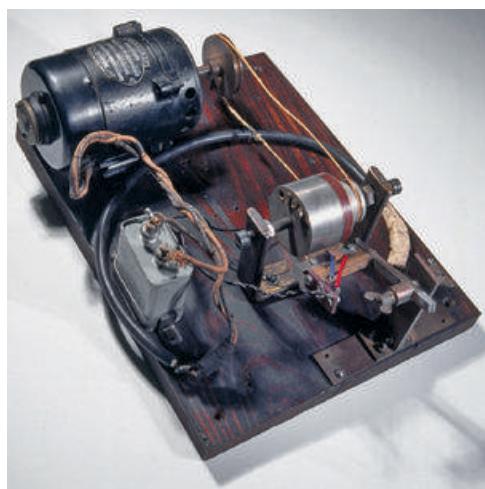
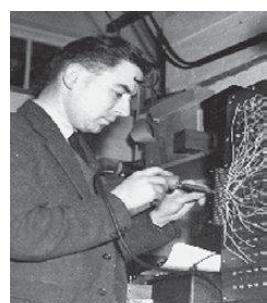


Fig. 2.25. Andrew Booth's magnetic drum memory.

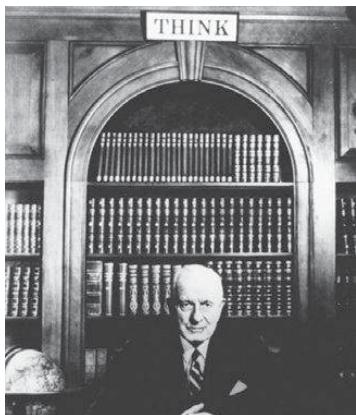


B.2.9. Andrew Booth (1918–2009), together with his assistant and future wife, Kathleen Britten, developed magnetic drum storage. He also invented a fast multiplication algorithm that is used in modern Intel microprocessors.



Fig. 2.26. Hard disk drive from IBM.

speeds of up to 15,000 rpm and the read and write heads operate within tens of nanometers of the surface. Originally introduced for IBM's main-frame computers, hard disks are now small enough to be incorporated in PCs, laptops, and even iPods. The company referred to hard disk drives as "direct access storage devices" or DASDs – rather than use the term *computer memory*. This was reportedly because Tom Watson Sr., the legendary first head of IBM, feared that an anthropomorphic term like *memory* might exacerbate people's fear and distrust of computers (B.2.10).



B.2.10. Tom Watson Sr. (1874–1956) had built IBM to be the dominant company in punched card tabulating machines that offered businesses and governments the ability to process huge amounts of data. IBM was also known for its highly effective salesmen dressed in ties and dark suits; for the company motto "THINK"; and the prohibition of any alcohol on IBM property. Watson is often credited with saying "I think there is a world market for maybe five computers," but there is no evidence that he actually said this. It was his son, Tom Watson Jr. (1914–93), who drove the company's move into electronic computers.

3 The software is in the holes

Computer programs are the most complicated things that humans have ever created.

Donald Knuth¹

Software and hardware

Butler Lampson, recipient of the Turing Award for his contributions to computer science, relates the following anecdote about software:

There's a story about some people who were writing the software for an early avionics computer. One day they get a visit from the weight control officer, who is responsible for the total weight of the plane.

"You're building software?"

"Yes."

"How much does it weigh?"

"It doesn't weigh anything."

"Come on, you can't fool me. They all say that."

"No, it really doesn't weigh anything."

After half an hour of back and forth he gives it up. But two days later he comes back and says, "I've got you guys pinned to the wall. I came in last night, and the janitor showed me where you keep your software."

He opens a closet door, and there are boxes and boxes of punch cards.

"You can't tell me those don't weigh anything!"

After a short pause, they explain to him, very gently, that the software is in the holes.²



Fig. 3.1. Carving holes on a punch card. The software in the early days of computing was represented by these holes but the cards were actually punched by machines.

It is amazing how these holes (Fig. 3.1) have become a multibillion-dollar business and arguably one of the main driving forces of modern civilization.

In the previous chapter we described the idea of separating the physical hardware from the software. Identifying these two entities is one of the key concepts of computer science. This is why we were able to go "down" into the hardware layers and see how the arithmetic and logical operations could be implemented without worrying about the software. Now let's go "up" into the software levels and find out how to tell the computer what to do. Software, also called a *program*, is a sequence of instructions which "give orders" to the hardware to carry out a specific task. In this program we can use only the operations

that the hardware can “understand” and execute. These core operations are called the “instruction set.” In terms of implementation the simplest instructions are hard-wired in the processor’s circuitry and more complex ones are constructed from the core instructions as we describe later in this chapter. For the program to work the instructions must be arranged in the right order and expressed in an unambiguous way.

A computer’s need for such exact directions contrasts with our everyday experience when we give instructions to people. When we ask a person to do something, we usually rely on some unspoken context, such as our previous knowledge of the person or the individual’s familiarity with how things work, so that we do not need to specify exactly what we want to happen. We depend on the other person to fill in the gaps and to understand from experience how to deal with any ambiguities in our request. Computers have no such understanding or knowledge of any particular person, how things work, or how to figure out the solution of the problem. Although some computer scientists believe that computers may one day develop such “intelligence,” the fact is that today we still need to specify exactly what we want the computer to do in mind-numbing detail.

When we write a program to do some specific task, we can assume that our hardware “understands” how to perform the elementary arithmetic and logic operations. Writing the precise sequence of operations required to complete a complicated task in terms of basic machine instructions at the level of logic gates would make for a very long program. For example, for numerical calculations we know we have an “add” operation in our basic set of instructions. However, if we have not told the machine how to carry out a multiply operation, it will not be able to do a simple calculation such as “multiply 42 by 3.” Even though for us it is obvious that we can multiply 42 by 3 by using multiple additions, the computer cannot make this leap of imagination. In this sense, the computer is “stupid” – it is unable to figure out such things for itself. In compensation, however, our stupid computer can add far faster than we can!

If we are trying to solve a problem that involves a numerical calculation requiring several multiplications, it will obviously simplify our program if we introduced a separate “multiply” instruction for the computer. This instruction will just be a block of additions contained in a program. Now when we give the computer the instruction “multiply 42 by 3,” the machine can recognize the word *multiply* and can start running this program. This ability to construct compound operations from simple ones and introduce a higher level of abstraction in this way is one of the fundamental principles of computer science. Moving up this ladder of abstraction saves us from having to write our programs using only the most elementary operations.

In the early days of computing, writing programs was not considered difficult. Even von Neumann thought that programming was a relatively simple task. We now know that writing correct programs is hard and error prone. Why did early computer scientists so badly underestimate the difficulty of this task? There are at least two possible explanations. The pioneers of computing were mostly male engineers or mathematicians who rarely spent time on the nitty-gritty details of actually writing programs. Like many men of the time, the early hardware pioneers thought that the actual coding of programs using binary

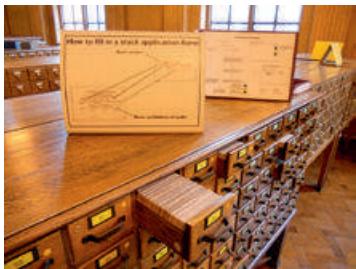


Fig. 3.2. A filing cabinet is a good analogy for thinking about the way a computer's hard disk stores information in files and folders.

numbers to represent the instructions and setting the appropriate switches on the machines was a secondary task, suitable for young women who needed only appropriate secretarial skills and some aptitude for mathematics. A more charitable explanation is that the difficulties of actually constructing the early computers and keeping them running were so overwhelming that these challenges completely overshadowed the problem of coding. Looking back, it now seems naive for the early computer builders to think that once they managed to get the machine up and running, the coding of a problem for the machine would be a relatively minor task!

There were some early warnings that programming might not be so simple. Maurice Wilkes recalls:

As soon as we started programming, we found out to our surprise that it was not as easy to get programs right as we had thought. I can remember the exact instant when I realised that a large part of my life from then on was going to be spent in finding mistakes in my own programs.³

In a similar vein, one of the pioneers of computing, Edsger Dijkstra of the Netherlands, suggests in his autobiography that programming is even harder than theoretical physics.

In 1955 I took the decision not to become a theoretical physicist, but to become a programmer instead. I took that decision because I had concluded that of theoretical physics and programming, programming embodied the greater intellectual challenge. You see, in those days I did not suffer from intellectual modesty.⁴

Software now surrounds us like the air that we breathe. It runs the communication networks, the power grid, our PCs, and our smart phones. It is embedded in cars, aircraft, and buildings, and in banks and national defense systems. We use software all the time, even if we are not aware of it. When we drive a car, pay bills, use a phone, or listen to a CD, thousands of lines of code are executed. According to Bjarne Stroustrup, the Danish inventor of the C++ programming language, “Our civilization runs on software.”⁵ Let’s take a closer look at how we write the software that controls such a large part of our lives.

The file clerk model

Computers do much more than just compute. Typically there will be one part of the machine where the computer does all the basic mathematical operations while the rest of the machine is dedicated to moving the digital data around in the form of electrical signals. In many ways, we can think of the operation of a computer as being like the work that file clerks once did in an office. The file clerk was given a job to do and then had to move back and forth to racks of filing cabinets, taking files out and putting them back, scribbling notes on pieces of paper, passing notes around, and so on, until the job was completed (see Fig. 3.2). It will be helpful to use this analogy of the computer as a file clerk as a starting point to explain some of the basic ideas of a computer’s structure and organization. This model is a good way for us to understand the essential ideas of how a computer actually does what it does.

Suppose there is a big company that employs many salespeople to sell its products and stores a great deal of information about the sales force in a big filing system consisting of cards in cabinets. Let's say that our file clerk knows how to get information out of the filing system. The data are stored in the filing cabinets on "Sales" cards, which contain the name, location, number of sales, salary, and other information about each salesperson. Suppose we want the answer to a simple question: "What are the total sales in the state of Washington?"

The instructions for the clerk could be:

Take out a "Sales" card.

If "Location" says *Washington*, then add the number of "Sales" to a running count called "Total."

Put the card back.

Take out the next "Sales" card and repeat.

This set of instructions looks fine, but what do we do if our file clerk does not know what is meant by keeping a "running count"? In this case we need to provide the clerk with more detailed instructions on exactly how to do that task. We therefore provide the clerk with a new card called "Total," and our more detailed "program" now reads:

Take out the next "Sales" card.

If "Location" says *Washington*, then take out the "Total" card.

Add the sales number to the number on the card.

Put the "Total" card back.

Put the "Sales" card back.

Take out the next "Sales" card and repeat.

In a modern computer, of course, the data would not be stored on cards and the machine does not physically take out a card. Instead, the computer reads the stored information from a *memory register*, a storage place in its main memory. Similarly, the computer can write from such a register to a "card" without actually physically taking out a card and putting it back.

To go any further with our analogy, we need to specify more precisely how our file clerk carries out the basic set of operations. One of the most elementary operations is that of transferring information from the cards that the clerk reads to some sort of scratch pad or working area where the clerk can do the arithmetic. We can do this by specifying exactly what our "Take" and "Replace" card operations mean:

"Take card X" means that the information on card X should be written on the scratch pad.

"Replace card Y" means that the information on the pad should be written on card Y.

We now need to instruct the clerk exactly how to check if the location on card X was *Washington*. The clerk will need to do this for each card, so he needs to remember *Washington* from one card to the next. One way to do this is to have *Washington* written on another card we shall call C. The instructions are now:

Take card X (from store to pad).

Take card C (from store to pad).

Compare what is on card X with what is on card C.

If the contents match, add the “Sales” to the “Total.” If not, replace both cards and take the next one.

It would obviously be more efficient not to keep taking out and putting back card C, the *Washington* card. If there is enough room on the scratch pad, the clerk could store this information on the pad for the duration of the calculation. This is an example of a trade-off in the hardware design: the balance between the clerk having to shuffle cards in and out versus increasing the amount of storage space needed on the pad. We can keep breaking down the clerk’s tasks into simpler ones until they correspond directly to basic operations that he knows how to carry out. For example, we need to tell the clerk precisely how to compare the information stored in “Location” and the name *Washington*.

Let us teach the file clerk how to use the scratch pad. The instructions can be divided into two groups. One group is a core set of simple procedures that come with the pad – add, transfer, and so on. In a computer, these instructions, called the *instruction set*, are the ones that have been implemented in the hardware: they do not change when we change the problem. They are like the clerk’s intrinsic abilities. Then there is a set of instructions specific to the task at hand, such as calculating the total number of sales for the state of Washington. This set of specialized instructions is the “program.” The program’s instructions can be broken down into operations from the core set as we have seen. The program represents the detailed instructions about how to use the clerk’s intrinsic abilities to do the specific job.

To get the right answer, the clerk must follow exactly the instructions that constitute the “program” in precisely the right order. We can ensure this by designating an area on the scratch pad to keep track of which steps have been completed. In a computer, this area is called a *program counter*. The program counter tells the clerk where he is in the list of instructions that constitute the program. As far as the clerk is concerned, this number is just an “address” that tells him where to look for the card with the instruction about what to do next. The clerk goes and gets this instruction and stores it on the pad. In a computer, this storage area is called the *instruction register*. Before carrying out the instruction, the clerk prepares for the next one by adding one to the number in the program counter. The clerk will also need some temporary storage areas on the pad to do the arithmetic, save intermediate values, and so on. In a computer, these storage areas are called *registers*. Even if you are only adding two numbers, you need to remember the first while you fetch the second. Everything must be done in the correct sequence and the registers allow us to organize things so that we achieve this goal.

Suppose our computer has four registers – A, B, and X plus a special register C that we use to store any number that we need to carry (put into another column) as part of the result of adding two numbers. Let us now look at a possible set of core instructions, the *instruction set*, for the part of a computer corresponding to the scratch pad. These instructions are the basic ones that will be built into the computer hardware. The first kind of instruction concerns the transfer of data from one place to another. For example, suppose we have

a memory location called M on the pad. We need an instruction that transfers the contents of register A or B into M or that moves the contents of M into register A or B. We will also need to be able to manipulate the program counter so we can keep track of the current number in register X. We therefore need an operation that can change this stored number as well as a “clear” instruction so that we can wipe out what was in a register and set it to zero. Then there are the instructions for the basic arithmetic operations, such as “add.” These instructions will allow us to add the contents of register B to the contents of register A and update the contents of register A with the sum A + B. We also need the logical operations: the logic gates AND and OR that allow the computer to make decisions depending on the input to these gates. This capability is important because it enables the computer to follow different branches of a program depending on the result of the logical operation. We therefore need to add another class of instructions that enable the computer to “jump” to a specific location. This instruction is called a *conditional jump*, an action in which the computer jumps to the new location in the program only if a certain condition is satisfied. This conditional jump instruction allows the machine to leap from one part of a program to another. And finally we will need a command to tell the computer when to stop, so we should add a “Halt” instruction to our list.

These elementary instructions now enable us to get the computer to do many different types of calculations. The machine is able to perform complex operations by breaking them down into the basic operations it understands. In the example above, our computer had only four registers. In modern computers, although the underlying concepts are exactly the same, a larger set of basic instructions and registers is typically built into the hardware. The lesson we should take from the file clerk analogy is this. As long as our file clerk knows how to move data in and out of registers and can follow a sequence of simple instructions using a scratch pad, he can accomplish many different complex tasks. Similarly, a computer does all of these tasks completely mindlessly, just following these very basic instructions, but the important thing is that it can do the work very quickly. As Richard Feynman (B.3.1) says, “The inside of a computer is as dumb as hell but it goes like mad!”⁶ A computer can perform many millions of simple operations a second, and multiply two numbers far faster than any human. However, it is important to remember that, at its heart, a computer is just like a very fast, dumb file clerk. It is only because a computer can do the basic operations so quickly that we do not realize that it is in fact doing things very stupidly.



B.3.1. Richard Feynman (1918–88) lecturing on computing at the workshop on Idiosyncratic Thinking at the Esalen Institute. In this lecture he explained in simple terms how computers work and what they are capable of. According to Feynman calling computers *data handlers* would be more fitting because they actually spend most of the time accessing and moving data around rather than doing calculations.

Maurice Wilkes and the beginning of software development

Let us now look at how the “file clerk model” is actually implemented in real computers. Computers work by allowing small electric charges to flow from the memory to collections of logic gates or to other memory locations. These flows are all regulated by von Neumann’s fetch-execute cycle: at each step, the computer reads an instruction from memory and performs the specified action. The section of the computer that does the actual computing is called the *processor* or the *central processing unit* (CPU). The processor has a

a graduate student in Cambridge, therefore introduced a program he called “Initial Orders” that read these more intuitive shorthand terms for the basic machine instructions, translated them into binary, and loaded them into memory ready for execution. Using these abbreviations made programs much more understandable and made it possible for users who were not computer specialists to begin to develop programs for the machine.

Wheeler’s Initial Orders program was really the first practical realization of von Neumann’s hardware-software interface. It was also the first example of an *assembly language*, a programming language that used names instead of numbers. Assembly languages constituted the next significant step up the hierarchy of software abstractions. They were easier to use than machine code but still much harder than later high-level languages. Writing an assembly language instruction such as “MOV R1 R2,” meaning “Move contents of register 1 to register 2,” allowed the user to avoid having to think about the explicit switches that needed to open to direct the flow of charge from register 1 to register 2. Similarly, *memory addresses*, the binary numbers that the computer used to track where data and instructions are stored in its memory, were replaced with more intuitive names like “sum” or “total.” Although assembly languages made the writing of programs easier, computers still ran with machine code, the low-level language composed of binary numbers. To run an assembly language program, the assembly language first has to be translated into machine code. The need for such translation required the development of programs called *assemblers* that could perform the translation and produce machine code as output. On the EDSAC, this translation was originally done with Wheeler’s Initial Orders program, which we might now call the first assembler.

Despite the great simplification introduced by using assembly language to write programs, the Cambridge team quickly found that a large amount of time was being taken up in finding errors in programs. They therefore introduced the idea of a “library” of tested, debugged portions of programs that could be reused in other programs. These blocks of trusted code are now called *subroutines*. For these blocks of code to be used in different programs, we need to use the *Wheeler jump*. As we have seen in our file clerk discussion, the computer takes its next instruction from the memory location specified in the special memory register called the *program counter*. After each instruction is read, the contents of the program counter are increased by one to point to the next instruction in memory. With a jump instruction, the computer can copy the memory address corresponding to the beginning of the subroutine code into the program counter. The computer is no longer restricted to the next instruction but can jump to the starting address of the subroutine code. The program will then follow the instructions in the subroutine code incrementing the program counter from that entry point. Of course, to know where in the program the subroutine should return after it has completed its execution, the computer also needs to have saved the previous contents of the program counter in another memory location, which was an essential feature of the Wheeler jump.

If we want to call a subroutine from within another subroutine, we will clearly need to save the multiple return addresses. We cannot use just one special memory location because the return address of the first subroutine will be



B.3.2. David John Wheeler
(1927–2004) was a British computing pioneer who made a major contribution to the construction and programming of the EDSAC computer. With his colleagues Maurice Wilkes and Stanley Gill, Wheeler invented *subroutines*, the creation of reusable blocks of code. Wheeler also helped develop methods of *encryption*, which puts a message into a form that can be read only by the sender and the intended recipient. Wheeler, Wilkes, and Gill wrote the first programming textbook in 1951, *The Preparation of Programs for an Electronic Digital Computer*.



Fig. 3.4. The information stored in a computer's memory is much like a stack of plates. Just as we can add or take plates only from the top of the stack, the last data added to memory must be the first removed.

overwritten by the return address of the second and so on. To overcome this difficulty, computer scientists introduced the idea of a group of sequential storage locations linked together to operate as a *memory stack* (Fig. 3.4). The memory stack functions much like a stack of dinner plates: plates can only be added or removed at the top of the stack. This is an example of a simple data structure that is useful in many applications. It is called a *LIFO stack*, which stands for Last In, First Out. This data structure is able to handle the storage of the return addresses of nested subroutines.

The conditional jump instruction introduces two powerful new concepts for programmers: *loops* and *branches*. With a conditional jump, the program only performs the jump if a certain condition is met; if not, the program continues down the original path. A loop is where a block of code is executed a specified number of times. This can be done by keeping a tally called a *loop count* that is increased by one on each pass through the code. At the end of each block of code, the loop count is tested to see if the specified number of repetitions has been carried out: if not, the program is sent back to the beginning of the loop. A branch is just what it says: the choice of what section of code to execute is made depending on the result of the condition.

In 1951, Maurice Wilkes, with David Wheeler and Stanley Gill, wrote up their experiences in teaching programming. Their book *The Preparation of Programs for an Electronic Digital Computer* was the first textbook on computer programming. Also in the same year, Wheeler was awarded the first PhD in computer science for his work with the EDSAC.

FORTRAN and COBOL: The story of John Backus and Grace Hopper

Although the computing fraternity was very much male-dominated in the early years, there were a few influential pioneers who were women. Probably the most famous is Grace Hopper, or Rear Admiral Professor Grace Hopper as she later became (B.3.3). Hopper received her PhD in mathematics from Yale University in 1934 and was teaching at Vassar College in Poughkeepsie, New York, when the United States entered World War II. She enlisted in the Naval Reserve in December 1943 and graduated at the top of her class in June 1944.

The Harvard Mark I of Howard Aiken had been commandeered for the war effort, and Aiken was now a Naval Reserve commander (B.3.4). He liked to say that he was the first naval officer in history who commanded a computer. Although Aiken's machine was not very influential on the future development of digital computers, Aiken was one of the first to recognize the importance of programming as a discipline. He persuaded Harvard to start the first master's degree courses in what would now be called computer science. In addition he insisted that the Mark I project be staffed with trained mathematicians. And this is how Lieutenant Grace Hopper of the U.S. Navy found herself being greeted by Aiken in the summer of 1944:

[Howard Aiken] waved his hand and said: "That's a computing machine." I said, "Yes, Sir." What else could I say? He said he would



B.3.3. Grace Hopper (1906–92), an American computer scientist, led the team that developed COBOL, the first programming language for business that allowed programmers to use everyday words.



B.3.4. Howard Aiken (1900–73), an American physicist and computer pioneer, proposed to IBM in 1937 the idea of constructing a large-scale electromechanical computer. By 1944, the computer, known as Mark I, became operational. The machine was fifty-five feet long, eight feet high, and about two feet wide.

like to have me compute the coefficients of the arc tangent series, for Thursday. Again, what could I say? “Yes, Sir.” I did not know what on earth was happening, but that was my meeting with Howard Hathaway Aiken.⁷

Hopper is credited with introducing the word *bug* into computer terminology after her team found a real insect, a moth, stuck in one of the machine’s relays (Fig. 3.5). At the end of the war, Hopper chose to remain at Harvard, programming first the Mark I and then the Mark II machines. But in 1949 she defected from Aiken’s camp and joined the opposition, the Eckert-Mauchly Computer Corporation in Philadelphia, which Aiken had dismissed as a foolish idea. This company was Eckert and Mauchly’s brave effort to commercialize their EDVAC ideas by building the UNIVAC computer. At the time, there was great skepticism as to whether there was a significant market for business computers. Hopper later joked:

Mauchly and Ekert had chosen their building perfectly. It was between a junk yard and a cemetery, so if it all went wrong ... they could throw the UNIVAC out of one window and themselves out of the other.⁸

Following along the same lines as Wilkes and Wheeler in Cambridge, Hopper and her team began to code their programs in UNIVAC assembly language. In doing this, she produced a program that would translate the assembly code into the binary machine code automatically, as Wheeler had done with his Initial Orders program. However, Hopper had much larger ambitions than using just primitive abbreviations for the low-level operations of assembly language. She began to investigate whether it was possible to write a program using expressions that more closely resembled everyday English. Initially focusing on creating a more natural language for scientific computing, Hopper experimented with a language called A-0. She introduced the term *compiler* for the software system she created to translate

Fig. 3.5. The first recorded appearance of a computer “bug” dates from 1947. Grace Hopper is credited with introducing the word into computer terminology after her team found a real insect, a moth, stuck in one of the relays of the Mark II computer. They removed the insect and taped it into the logbook.

9/9	
0800	Antron started
1000	stopped - antron ✓ { 1.2700 9.032 847 025 13.025 (025) MP - NC 9.037 846 995 const 023 PRO 2 2.13097 GY5 convd 2.13097 GY5
	Rely's 6-2 in 023 failed special speed test in relay 11.000 test.
1100	Rely's changed
1525	Started Cosine Tape (Sine check) Started Multi Adder Test.
1545	
16500	FIRST actual case of bug being found.
1700	antron started. closed down.

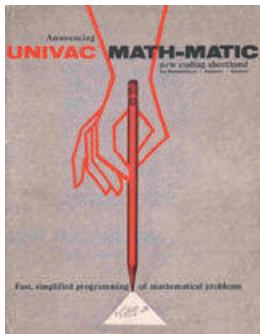


Fig. 3.6. In 1955, Hopper and her team released the MATH-MATIC language for the UNIVAC. MATH-MATIC was one of the first higher-level languages above assembly language to be developed.

programs written in A-0 to UNIVAC machine code. The results, she reported, were mixed:

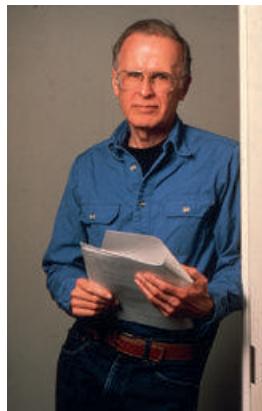
The A-0 compiler worked, but the results were far too inefficient to be commercially acceptable. Even quite simple programs would take as long as an hour to translate, and the resulting programs were woefully slow.⁹

However Hopper remained an energetic advocate for *automatic programming*, in which the computer generates the machine code from a program written in a “high-level” programming language. She and her team therefore persevered with the development of the A-0 language and its compiler. In 1955, the company released the MATH-MATIC language for the UNIVAC (Fig. 3.6), and a news release declared, “Automatic programming, tried and tested since 1950, eliminates communication with the computer in special code or language.” These attempts in the early 1950s had shown that the outstanding problem in programming technology was to produce a compiler that could generate programs as good as those written by an experienced assembly language or machine code programmer. Enter John Backus and IBM (B.3.5).

IBM introduced its 704 computer for scientific applications in 1954. A major advance in the architecture of the 704 was that the hardware included dedicated circuits to perform the operations needed to handle *floating-point numbers* – numbers containing fractions in which the decimal point is moved to a standard position in order to simplify the hardware required to manipulate such fractional numbers – and not just integers (whole numbers). Now programmers could add, subtract, multiply, and divide real numbers as easily as performing these same operations with integers, without having to call on complex subroutines to do these operations. Backus had been developing an assembly language for another IBM computer, but in late 1953 he sent a proposal to his manager suggesting the development of what he called a “higher level language” and compiler for the IBM 704. It is interesting that Backus made the case for such a language mainly on economic grounds, arguing that “programming and debugging accounted for as much as three-quarters of the cost of operating a computer; and obviously as computers got cheaper, this situation would get worse.”¹⁰

The project was approved, and the FORTRAN – for FORmula TRANslation – project began in early 1954. Producing code that was nearly as good as that written by an experienced machine code programmer was always the overriding goal of Backus’s team:

We did not regard language design as a difficult problem, merely a simple prelude to the real problem: designing a compiler which could produce efficient [binary] programs. Of course one of our goals was to design a language which would make it possible for engineers and scientists to write programs for the 704. We also wanted to eliminate a lot of the bookkeeping and detailed repetitive planning which hand coding [in assembly language] involved.¹¹



B.3.5. John Backus (1924–2007), a computer scientist at IBM, developed FORTRAN, the first programming language that enabled scientists and engineers to write their own programs.

In April 1957, the language and the compiler were finished. The compiler consisted of about twenty thousand lines of machine code and had taken a team of about a dozen programmers more than two years to produce.

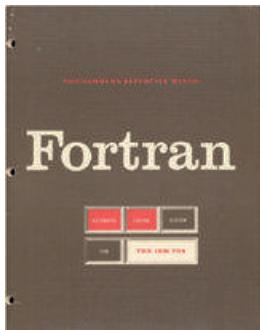


Fig. 3.7. The first FORTRAN book from IBM.



Fig. 3.8. FLOW-MATIC, developed by Grace Hopper in 1956, was the first programming language that allowed users to describe operations in English-like expressions.



Fig. 3.9. COBOL, one of the first programming languages, is still alive and well. Much business transaction software is written in COBOL and there are about 200 billion lines of code in use. Ninety percent of financial transactions are written in COBOL.

A typical statement in FORTRAN looks very like the underlying mathematical equation. Thus

$$y = e^x - \sin x + x^2$$

became

$$y = \text{EXP}(X) - \text{SINF}(X) + X^{**2}$$

Because of this simplicity and how closely it resembles the language of mathematics, FORTRAN rapidly became the dominant language for scientific computing. Backus's team had come very close to meeting their design goal:

In practice the FORTRAN system produced programs that were 90% as good as those written by hand, as measured by the memory they occupied or the time they took to run. It was a phenomenal aid to the productivity of a programmer. Programs that had taken days or weeks to write and get working could now be completed in hours or days.¹²

FORTRAN also produced another great benefit – the portability of programs across different machines. Although the first compiler was written for the IBM 704, very soon there were FORTRAN compilers for other IBM computers. Competing computer manufacturers also soon produced FORTRAN compilers for their machines. For the first time there were computers capable of speaking the same language so that programmers did not have to learn a new language for every new computer.

In 1961, Daniel McCracken published the first FORTRAN programming textbook for use in undergraduate courses in universities. In 1966, FORTRAN became the first programming language to be formally standardized by ANSI, the American National Standards Institute, the organization that creates standards for the U.S. computer industry (Fig. 3.7). The FORTRAN language, now written *Fortran* with only one capital letter, has evolved with time to incorporate new structures and technologies from research on programming languages by computer scientists. It is nevertheless surprising that Fortran programs are still much used in scientific computing more than fifty years after the first introduction of the language.

The other major breakthrough in early computer programming was a language for business applications. After her work on MATH-MATIC, Hopper turned to the problem of making business programming – the tasks needed to run a business such as managing accounting and inventory – easier and more intelligible to that community. By the end of 1956, she had produced a compiler for FLOW-MATIC, a language that contained around twenty English-like expressions and allowed the use of long character names (Fig. 3.8). For example, to test whether the value of variable A is greater than that of variable B, in Fortran we would write:

IF A.GT. B

By contrast, in a language like FLOW-MATIC, one would write a similar comparison test as:



Fig. 3.10. Early computers used punched cards to input programs and data.

IF EMPLOYEE-HOURS IS GREATER THAN MAXIMUM

The language not only made the programs more intelligible to managers but also provided a form of self-documentation that describes what the program is supposed to do.

In May 1959 the U.S. Department of Defense started an initiative to develop a common business language. This led to the COBOL (Fig. 3.9) programming language – for COmmon Business-Oriented Language – which was strongly influenced by Hopper’s earlier FLOW-MATIC language. For this reason, Hopper is sometimes known as “the mother of COBOL.” What made the language so successful was a declaration by the U.S. government a year later that it would not lease or purchase any new computer without a COBOL compiler. At the end of 1966, Hopper retired from the Navy with the rank of commander. Less than a year later she was recalled to active duty and tasked with the job of rewriting the Navy’s payroll system in COBOL. She was promoted to rear admiral in 1985.

For the next twenty years, from about 1960 to about 1980, FORTRAN and COBOL accounted for approximately 90 percent of all applications programs. Backus went on to develop a notation to capture the “grammar” of a programming language – that is, the way in which the special words and concepts of a language can be put together. A Danish computer scientist, Peter Naur, then simplified Backus’s notation so that the grammar of any language could be captured in what is now known as Backus-Naur Form or BNF (B.3.6). In the 1970s Bell Labs produced a *compiler-compiler*, a program that could transform a BNF specification into a compiler for that language. There has been much research and experimentation with programming in the fifty years since FORTRAN and COBOL. We will look at some of these developments in the next chapter.

Early operating systems

In using even these early machines, it clearly made no sense for each user to have to figure out independently how to interact with the computer. Originally, users might input their programs and data – send instructions and information to the computer – using a punched card or paper tape reader. Later the input process might involve a keyboard, mouse, or, nowadays, a touch-enabled tablet. Each user could also use disk drives to access and store data, and could read off the results from a printer or some form of screen display. So although the earliest computers had no real *operating system* – that is, no software to control the operation of the entire computer system with anything like the sophistication we see today – it was still useful to collect together all the I/O subroutines – programs for input and output – and have them permanently loaded on the machine.



B.3.6. Peter Naur, a Danish computer scientist, helped develop a successful programming language called Algol 60. In 2005, Naur received the Turing Award for his contributions to computer science.

In the earliest days of computers, users had to book a time slot on the machine, so graduate students were naturally allocated the nighttime slots! Users loaded their programs into the machine using punched cards or paper tape and then waited while the computer ran their program (Fig. 3.10). This personalized system quickly evolved to a more efficient system in which the users were isolated from the machine by “operators.” Users now had to give their program deck to the operator, who would load a batch of such programs

The Computing Universe

into the machine and then return the output to the users when the jobs were completed. The “operating system” was just the loading routine used by the operator to schedule the jobs on the computer plus the collection of I/O subroutines. As commercial computers began to appear in the early 1950s, such *batch processing* was the norm.

By the mid- to late 1950s, the limitations of batch processing were becoming apparent, and in universities there was a great deal of experimentation with the idea of more interactive computing. In 1955, John McCarthy (B.3.7), one of the pioneers of *artificial intelligence*, spent a summer at IBM’s laboratory in Poughkeepsie and got to learn computer programming through batch processing on the IBM 704 computer. He was appalled at having to wait to learn whether or not his program had run correctly. He wanted the ability to debug the program interactively in “real time,” before he had lost his train of thought. Because computers were very expensive systems at that time, McCarthy conceived of many users sharing the same computer at one time instead of just being allowed access to the machine sequentially, as in batch processing. For such sharing to be possible, multiple users had to be connected to the machine simultaneously and be assigned their own protected part of memory for their programs and data. Although there was only one CPU, each user would have the illusion that he or she had sole access to it. McCarthy’s idea was that because the computer cycles from instruction to instruction very quickly on a human timescale, why not let the CPU switch from one memory area and program to another memory area and program every few cycles? This way the user would have the illusion that they are sole user of the machine. He called his concept *time sharing*:

Time-sharing to me was one of these ideas that seemed quite inevitable.
When I was first learning about computers, I [thought] that even if [time sharing] wasn’t the way it was already done, surely it must be what everybody had in mind to do.¹³



B.3.7. John McCarthy (1927–2011) contributed many groundbreaking ideas to computing, such as time sharing, the LISP programming language, and artificial intelligence. In recognition of his pioneering work in computer science, he received the Turing Award in 1971.



B.3.8. Fernando Corbató pioneered the development of operating systems that allowed multitasking and time sharing. He stated a rule of computer science called Corbató’s law, according to which “The number of lines of code a programmer can write in a fixed period of time is the same independent of the language used.”

Time sharing was not what IBM had in mind. It is perhaps understandable that IBM had little interest in time sharing and interactive computing, despite its longtime involvement in postwar projects with MIT, because all of its business customers were happy with their new batch-mode IBM computers. In order to implement time sharing, McCarthy needed IBM to make a modification to the hardware of the 704. This was needed for an “interrupt” system that would allow the machine to suspend one job and switch to another. Fortunately IBM had created such an interrupt modification for the Boeing Company to connect its 704 computer directly to data from wind-tunnel experiments. IBM allowed MIT to have free use of the package and in 1959 McCarthy was able to demonstrate an IBM 704 computer executing some of his own code between batch jobs. In his live demonstration, the time-sharing software was working fine until his program unexpectedly ran out of memory. The machine then printed out the error message:

THE GARBAGE COLLECTOR HAS BEEN CALLED. SOME INTERESTING STATISTICS ARE AS FOLLOWS ...¹⁴

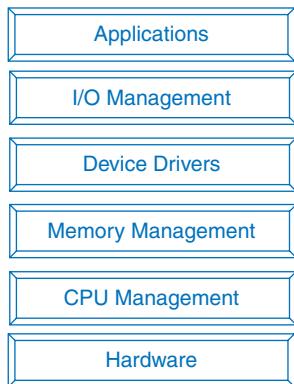


Fig. 3.11. A computer's operating system can be pictured in layers. The bottom layer consists of *hardware*, the mechanical, electronic, and electrical parts that make up the machine. Higher layers represent the main functions of the operating system, including the CPU that does the actual computing, management of the computer's memory, and device drivers that operate devices attached to the computer. Higher still is the I/O layer, which enables users to communicate with the computer. At the top are the applications that perform specific tasks, such as word processing and managing databases.

McCarthy's audience at MIT thought he had been the victim of a practical joke. In fact he was writing his programs in LISP (List Processing), a high-level language he had developed for programming artificial intelligence applications. For this language, he had introduced a *garbage collection* routine to reclaim parts of the memory that were no longer needed by the program. In effect, McCarthy's routine was an early attempt to build an automatic memory management system.

It was not until 1961 that Fernando Corbató ([B.3.8](#)) at the MIT Computation Center was able to demonstrate a fully working time-sharing system. This was called the Compatible Time-Sharing System or CTSS. This was the starting point for J. C. R. Licklider's famous Project MAC, a time-sharing system of which the goal was nothing less than what its proponents called "the democratization of computing." The MAC project (MAC could stand for Machine-Aided Cognition or Multiple Access Computer) and the Multics (Multiplexed Information and Computing Service) time-sharing operating system that developed from these beginnings were enormously influential and led to spin-off projects in many different areas. Most modern operating systems use an interrupt system to shift resources when and where they are needed, making multitasking possible.

The many roles of an operating system

Operating systems have progressed a long way from being a simple collection of subroutines and a batch loader to software systems of enormous complexity and power. We end this chapter by listing the major functions that a modern operating system must carry out.

Device drivers and interrupts

One of the earliest roles of the operating system was to allow users to interact with a wide variety of devices, such as keyboards, scanners, printers, disks, and mice without having to write their own code. The key to making this possible with all the multitude of different devices we have today is to hide all the intricate details of a particular device behind a standard piece of software called a *device driver*, a program that operates a particular type of device attached to the computer. The interface of a device driver with the computer needs to be carefully specified because many devices need to access specific memory locations. They also must generate and respond to control signals called *interrupts*, indications that some event happening in the computer needs immediate attention. Handling these interrupts is a key function of the operating system ([Fig. 3.11](#)).

Job scheduling

If one program has to wait for some input, another program could start running. The operating system must have the capability of sending a waiting program to "sleep" and then waking it up again with an interrupt when its input has arrived and it is ready to proceed. To do this, the operating system needs to maintain a table of "active processes," the operations that are under way. This list contains all the details for each process - where it is in main memory,

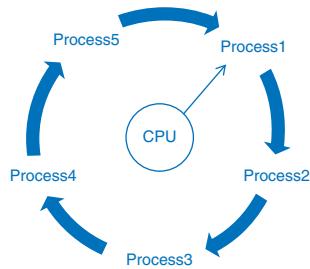


Fig. 3.12. Perhaps the most important task for an operating system is allocating time on the CPU to different processes. Each process is allowed to use the CPU for a limited time.

what the current contents of the CPU registers are, what addresses are in the program counter and the memory stack, and so on. When a process becomes active, the operating system loads all of this information into the CPU and restarts the program from where it left off. The operating system also needs to have some “scheduling policy” to decide which should be the next process to become active. There are many such scheduling policies that attempt to ensure “fair” process selection – but most users would argue that none of them are perfect!

Hardware interrupts

The next problem for the operating system is fundamental. Because the operating system is also a program and the CPU can only run one program at a time, how can the operating system hand the CPU over to one process and then get back control of the CPU so it can schedule another process? This is done by a different type of interrupt, one that switches not between a process and the operating system but between the actual computer hardware and the operating system. It is called a *hardware interrupt*. Such an interrupt happens when some event like a keyboard entry or mouse movement occurs. The hardware interrupt changes the computer’s operation from *user mode* to *supervisor mode*. Supervisor mode is a method of operation in which the operating system can access and use all of the computer’s hardware and instructions. By contrast, in user mode only a restricted set of hardware and instructions are accessible to the program. At a hardware interrupt, the computer jumps to the *scheduler* program. This software finds out what event has happened and decides which user process should next gain control of the CPU (Fig. 3.12).

System calls

In addition to hiding the complexity of devices through standard interfaces with device drivers (Fig. 3.13) and scheduling user processes, the operating system manages how programs request services from the hardware. When user programs need to access and control devices directly, the role of the operating system is to make sure they do so safely without causing damage to the hardware. The operating system ensures the safety of the entire computer system through a set of special-purpose functions called *system calls*. System calls are the means by which programs request a service from the operating system.

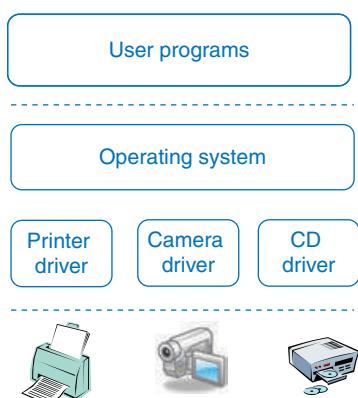
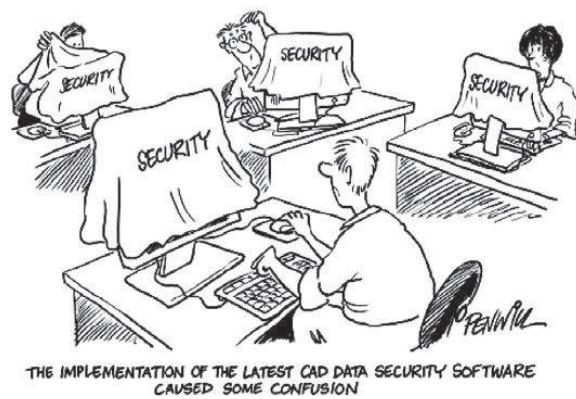


Fig. 3.13. A significant portion of the code of an operating system is made up of device drivers. These are programs that operate various devices attached to the computer, such as a printer, a camcorder, or a CD player.

File management

One special class of system calls has come to symbolize the entire operating system. These are the calls that create and manipulate the file system. Computers store data in a hierarchical arrangement of *files* or *folders* that are accessed through “directories” on a hard disk. The *hard disk*, housed in a unit called a hard drive consists of magnetic plates called *platters* that store information. The computer reads and writes data to the disk. The operating system has to keep track of the file names and be able to map them to their physical location on the disk.

Fig. 3.14. With the widespread use of the Internet, online security is becoming one of the most important aspects of present-day computing.



Virtual memory

Besides managing the file store, where the files are kept on the hard disk, the operating system also manages the computer's main memory. Computer memory is expensive, and typically a computer has much less main memory than its address space can support. A computer's *address space* represents the range of numbers that can be used for addressing memory locations. For example, if the CPU registers are 32 bits wide, they can hold 2^{32} different bit patterns. This is the largest possible address space and is usually referred to as 4 gigabytes, because 2^{32} is 4,294,967,296, or just more than four billion (a thousand million). G stands for *giga*, a prefix that means one billion. Nowadays users can write programs without worrying about the limitations of main memory. Clever virtual memory software allows user programs to assume they can employ all of the addressable memory even though the main memory supports far fewer real addresses. The virtual memory creates this illusion by moving blocks of memory called "pages" back and forth between the hard disk and the main memory. This leads to a new type of interrupt called a *page fault*, which occurs when the page that the program needs is not yet in main memory. The operating system then must suspend the program so that the required page can be copied into main memory from the hard disk. To make room for this page in main memory, another page must be swapped out. Memory mismanagement is one of the most common causes of "crashes" and there are many elaborate strategies for deciding which page is best to move out.

Security

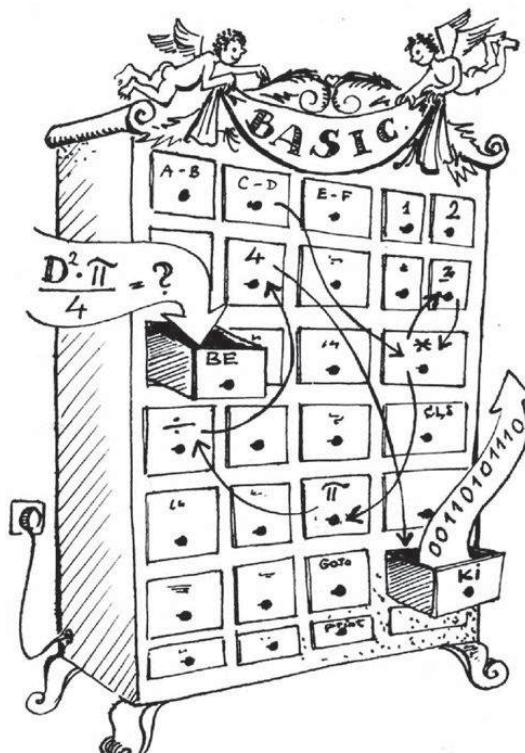
One vital function that an operating system must provide is security (Fig. 3.14). For each user the operating system must maintain the confidentiality and integrity of the information they have stored on the computer. A first step toward this goal is to identify permitted users by a password so that they must use this password to log into the computer before they are allowed access to any of its resources. The operating system must keep track of the users and passwords and ensure that only those files associated with an authorized user can be accessed and manipulated by that user. Alas there is now a thriving subculture of *hackers*, skilled programmers who try to subvert these security

measures. Suppliers of operating systems are locked into an escalating struggle to develop effective countermeasures to foil hackers.

We will return to the problem of hackers later in this book when we come to the creation of the Internet and the personal computer. In the next chapter we look at the continued development of programming languages and of the attempt to turn the business of writing programs into a real engineering discipline.

Key concepts

- Instruction set
- File clerk model of computer
- Machine code and assembly language
- Subroutines, loops, and branches
- FORTRAN and COBOL
- Operating system concepts
 - Batch processing and time sharing
 - Device drivers
 - Interrupts
 - System calls
 - Memory management
 - Security
- Microcode



Microcode

In 1951, at a ceremony to inaugurate the Manchester University computer, Maurice Wilkes argued that “the best way to design an automatic calculating machine”¹⁵ was to build its control section as a stored-program computer of its own. Each control operation, such as the command to add two numbers, is broken down into a series of *micro-operations* that are stored in a *microprogram*. This method had the advantage of allowing the hardware design of the control unit to be simplified while still allowing flexibility in the choice of instruction set.

The microcode approach turned out to be the key insight that enabled IBM to successfully implement its ambitious “360” project (Fig. 3.15). The 360 project was an attempt by IBM in the mid-1960s to make all IBM computers compatible with one another, simplifying what many people saw as the confusing tangle of the company’s machines. Thomas Watson Jr. (B.3.9), then president of IBM, set up a group called the SPREAD (Systems Programming, Research Engineering and Development) Committee to investigate how this goal could be achieved. At one stage, two key architects of the System/360 family of mainframe computers, Fred Brooks and Eugene Amdahl, argued that it couldn’t be done. However, an English engineer called John Fairclough (B.3.10), who was a member of SPREAD, had studied electrical engineering at Manchester and learned about the advantages of Wilkes’s microprogramming and microcode. It was through him that IBM realized that microcode offered a solution to the problem of offering a common instruction set across the System/360 family of computers. Microcode also gave engineers the possibility of offering *backward compatibility*, which would enable a new computer to run the same software as previous versions of the machine. By installing microcode that implemented instructions written for programs developed for earlier machines, the older programs would still be able to run on the new 360 computer. Fairclough later became director of IBM’s UK Development Laboratory at Hursley, near Winchester (Fig. 3.16).



Fig. 3.15. The IBM System/360 was a family of general-purpose mainframe computers delivered in 1965. It was the first line of computers designed to be compatible with one another.



B.3.9. Thomas Watson Jr. (1914–93), then president of IBM, took an unprecedented gamble by putting huge resources into the IBM 360 project to unify IBM’s many different computing systems. The gamble paid off and changed the history of computing. From 1979 to 1981, Watson served as the U.S. ambassador in Moscow.



Fig. 3.16. IBM Hursley Laboratories near Winchester, U.K. The Lab developed several IBM computers and much important software. The software produced by Hursley includes one of the best-selling software products of all-time, CICS, for transaction processing – the day-to-day transactions of banking, airline ticket systems, and so on.



B.3.10. John Fairclough (1930–2003) played an important role in the British computing industry. He was a member of the IBM 360 team and, in 1974, he became managing director of IBM Hursley Laboratory near Winchester in England. In the 1980s, Fairclough served as chief scientific adviser to the British government. He strongly supported close collaboration between universities and computer designers and manufacturers.

4 Programming languages and software engineering

The major cause of the software crisis is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.

Edsger Dijkstra¹

The software crisis

The term *software engineering* originated in the early 1960s, and the North Atlantic Treaty Organization sponsored the first conference on the “software crisis” in 1968 in Garmisch-Partenkirchen, West Germany. It was at this conference that the term *software engineering* first appeared. The conference reflected on the sad fact that many large software projects ran over budget or came in late, if at all. Tony Hoare, a recipient of the Turing Award for his contributions to computing, ruefully remembers his experience of a failed software project:

There was no escape: The entire Elliott 503 Mark II software project had to be abandoned, and with it, over thirty man-years of programming effort, equivalent to nearly one man’s active working life, and I was responsible, both as designer and as manager, for wasting it.²

In his classic book *The Mythical Man-Month*, Fred Brooks (B.4.1) of IBM draws on his experience developing the operating system for IBM’s massive System/360 project. Brooks makes some sobering reflections on software engineering, saying, “It is a very humbling experience to make a multimillion-dollar mistake, but it is also very memorable.”³

In this chapter we will explore two aspects of the way the software industry has addressed this crisis: the evolution of programming languages (Fig. 4.1) and the emergence of software engineering methodologies. We will see how two major ideas provide the basis for modern software development: (1) *structured programming*, in which the statements are organized in a specific way to minimize error; and (2) *object-oriented software*, which is organized around the



B.4.1. Fred Brooks made a major contribution to the design of IBM/360 computers. His famous book *The Mythical Man-Month* describes his experiences in software development.

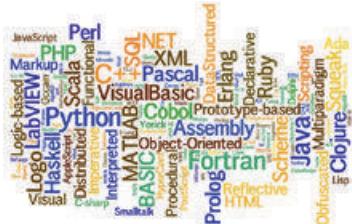


Fig. 4.1. Some popular programming languages. Wikipedia lists more than seven hundred programming languages but only a few of these have attained widespread use.

objects the programmer wants to manipulate rather than the logic required to do individual operations. To computer scientists, an *object* is any item that can be individually selected and handled. In object-oriented programming, an *object* consists of not only the data but also the *methods* employed to operate on that type of data. In addition, the introduction of engineering practices in the specification, design, coding, and testing of software has helped the software industry make progress toward taming the software crisis. However, it is important to remember that even with the best software engineering practices of today, software systems contain somewhere between ten and ten thousand errors per million lines of code. It is therefore not surprising that testing and bug fixing play a large role in the life cycle of software. Lastly, we shall take a look at an alternative model of software development. This model is based on *crowdsourcing*, which incorporates contributions from a large group of people, and the free sharing of the *source code*, the program instructions in their original form. Such sharing of source code is called the *open-source* approach. One of its major advantages is that it permits rapid finding of bugs. Eric Raymond has memorably summarized this idea as “Given enough eyeballs, all bugs are shallow.”⁴ In other words, the more people who see and test a set of code, the more likely any flaws will be caught and fixed quickly. Raymond called this “Linus’s Law” after Linus Torvalds, creator of the open-source Linux operating system. However, other experienced software engineers would contest this statement!

Elements of modern programming languages

Before the introduction of FORTRAN in the 1950s, programmers had to work in machine code, made up of binary commands (Fig. 4.2), or in assembly languages, composed of symbolic expressions, both of which were difficult and time-consuming to write. FORTRAN was the first commercially successful *high-level language*. That is, it resembled natural human language, was easy to learn, and required less knowledge of the computer on which it would be run. Many people expected that FORTRAN would make software easy to create and simple to fix. It is amusing to look back at the optimism expressed in the original FORTRAN proposal from John W. Backus in 1954:

Since FORTRAN should virtually eliminate coding and debugging, it should be possible to solve problems for less than half the cost that would be required without such a system. Furthermore, since it will be possible to devote nearly all usable machine time to problem solution instead of only half the usable machine time, the output of a given machine should be almost doubled.⁵

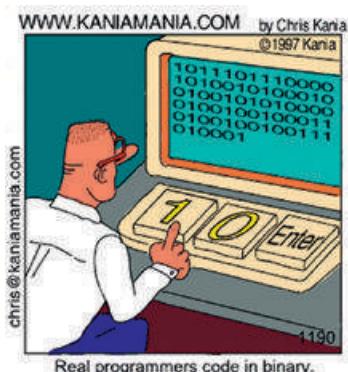


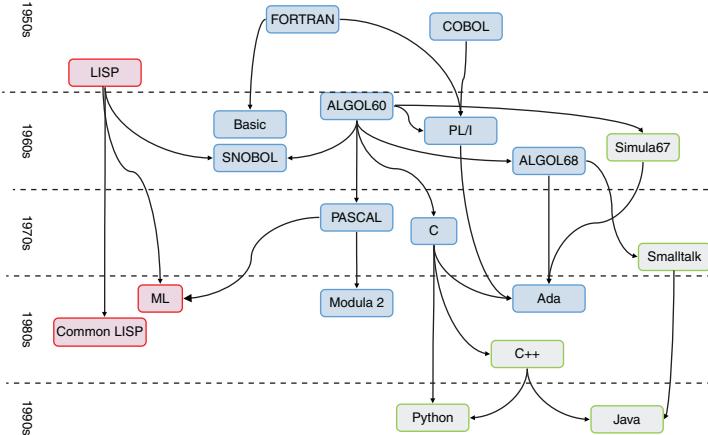
Fig. 4.2. In the early days of computing, programming was done in binary. It was only with the introduction of assembly language that programming became slightly more intuitive. High-level programming languages like FORTRAN and COBOL only became popular with the development of efficient compilers that produced performance close enough to assembly language programs.

So how did software production reach such a crisis point only a decade later? Part of the answer lies in the features of early languages.

We will begin by looking at some early programming languages and three concepts that led to better programming: (1) *type checking*, which checks and enforces certain rules to help prevent one major category of errors; (2) *recursion*, where a function can call itself; and (3) *dynamic data structures*. A *data structure* is any organized form, such as a list or file, in which connected data items are held in a computer. A dynamic data structure provides a way of organizing data that allows more efficient use of computer memory.

The Computing Universe

Fig. 4.3. A simplified evolutionary graph of popular programming languages. The language styles are marked by different color frames: red – declarative, blue – imperative (procedural), and green – object oriented.



Programming languages are artificial languages designed to communicate instructions to a computer. FORTRAN and most other programming languages consist of sequences of text, including words, numbers, and punctuation marks. Programming languages may be divided into two broad categories, *imperative* and *declarative* languages (Fig. 4.3). Roughly speaking, imperative languages specify *how* a computation is to be done, while declarative languages focus on *what* the computer is supposed to do. FORTRAN was the first commercial example of an imperative language. A year later, in 1958, John McCarthy and his students at the Massachusetts Institute of Technology (MIT) developed LISP, standing for LISt Processing, a programming language targeted at artificial intelligence (AI) applications. LISP was the first attempt at creating a type of declarative language in which computation proceeds by evaluating functions.

In computer programming, a *variable* is a symbol or name that stands for a location in the computer's memory where data can be stored. Variables are important because they enable programmers to write flexible programs. Rather than putting data directly into the program, the programmer can use variables to represent the data, allowing the same program to process different sets of information, depending on what is stored in the computer's memory. Instructions known as *declaration statements* specify the sort of information each variable can contain, called the *data type*.

Many of the main ideas of the original FORTRAN language are still used in programming languages today. Features of early versions of FORTRAN were:

- Variable names in a FORTRAN program could be up to six characters long and could change their values during execution of the program.
- Variable names beginning with the letters I, J, K, L, M, or N represented *integers*, that is, whole numbers with no fractional parts. All other variables represented *real numbers*, which could be any positive or negative numbers, including integers.
- Boolean variables, variables that have the value of either true or false, could be specified with a *logical declaration statement*.
- Five basic arithmetic operations were supported: + for addition; - for subtraction; * for multiplication; / for division; and ** for exponentiation, raising one quantity to the power of another.

Another important feature of FORTRAN was that it introduced a data structure called an *array* that was especially useful for scientific computations. An array is a group of logically related elements stored in an ordered arrangement in the computer's memory. Individual elements in the array may be located using one or more indexes. The *dimension* of an array is the number of indexes needed to find an element. For example, a list is a one-dimensional array, and a block of data could be a two- or three-dimensional array. An instruction called a *dimension statement* instructs the compiler to assign storage space for each array and gives it a symbolic name and dimension specifications. For example, a one-dimensional array of numbers called a *vector* may be specified by the dimension statement VEC(10) or a three-dimensional field of values by MAGFLD(64, 64, 64).

Modern programming languages have improved on the minimal specification of data types in FORTRAN. Many languages today employ *strong typing* – strict enforcement of type rules with no exceptions. Checking that a large, complex software system correctly uses appropriate data types greatly reduces the number of bugs in the resulting code.

For efficiency reasons, the development of FORTRAN was closely aligned to the hardware architecture of the computer. Thus FORTRAN *assignment statements*, instructions that assign values to the variables, are actually descriptions of how the data moves in the machine. In FORTRAN, assignment statements are written using the “equal” sign, but these statements do not mean mathematical equality. Consider the assignment statement

$$A = 2.0 * (B + C)$$

In FORTRAN, this statement means, “Replace the value at address A by the result of the calculation $2.0*(B+C)$.” Similarly, the odd-looking statement

$$J = J + 1$$

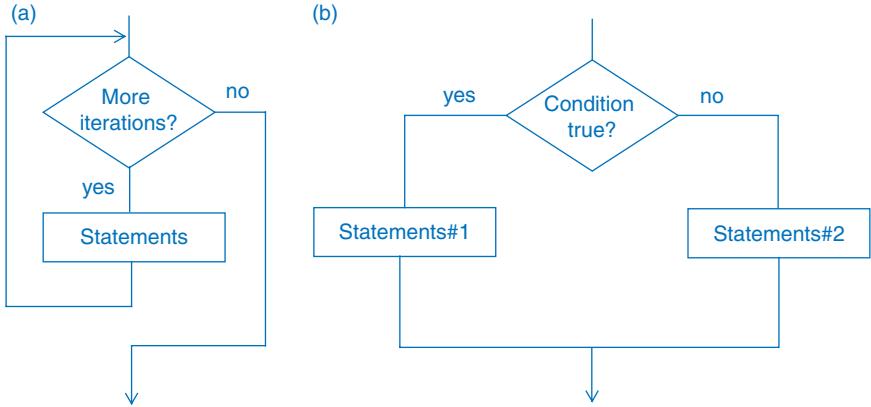
means, “Read the value from address J, add 1 to this value, and then store the result at the same address.”

The original FORTRAN specification provided three kinds of *control statements*, instructions that switch the computer from automatically reading the next line of code to reading a different one:

- The DO statement was used to allow the program to perform a *loop*, a sequence of instructions that repeats either a specified number of times or until a particular condition is met.
- The IF statement was employed for *branches*, instructions that tell the computer to go to some other part of the program, depending on the result of some test.
- The GO TO statement was used to direct the computer to jump to a specific numbered statement.

Forms of the basic *do* loop and *if* statement constructs are available in most modern programming languages. We can write the general form of these control statements in *pseudocode*, an informal, high-level description of a program that is intended to be read by humans. A computer could not directly execute

Fig. 4.4. Flowcharts for the “for” loop (a) and “if-then-else” (b). They illustrate the decision points in the program where a branch can be made depending on a check of some condition. The “for” loop performs a set number of iterations through the same piece of code that use different data. The “if-then-else” branch executes a different piece of code depending on the result of a test condition.



it, however, because the pseudocode version of the program omits details not essential for understanding by people but necessary for a machine to run the program. Thus we can write a *do* loop using the *for* keyword

```
for <var> in <sequence>
    <statements>
```

The variable *<var>* after the keyword *for* is called the *loop index*, and the statements in the body of the *for* loop – *<statements>* – are executed according to the number of times specified in the *<sequence>* portion of the loop heading. After completing the loop the required number of times, the program goes to the program statement after the last statement in the body of the loop.

Similarly, we can write an *if* statement as:

```
if <condition>:
    <statements#1>
else:
    <statements#2>
```

If the Boolean condition *<condition>* is true, the program executes the first set of statements – *<statements#1>* – and then jumps to the statement after the second block of statements – *<statements#2>*. If the condition is false, the program jumps over the first block of statements and executes the second block of code *<statements#2>* under the *else* keyword, before carrying on with the next statement of the program. It is sometimes helpful to visualize these control structures as diagrams called *flowcharts*, first introduced into computing by Herman Goldstine and John von Neumann in 1947. Flowcharts use boxes to represent the different parts of the program with arrows between the boxes showing the sequence of events. Flowcharts representing the *for* loop and the *if-then-else* flowcharts are shown in Fig. 4.4.

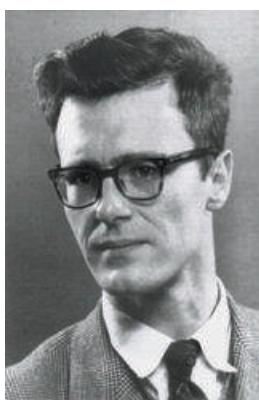
Modern programming languages generally do not encourage use of the *go to* statement. Undisciplined use of *go to* statements often led to very complex “spaghetti” code that was difficult to understand and debug. In 1968, Edsger Dijkstra (B.4.2) wrote a famous article titled “Go To Statement Considered Harmful”:

For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of go to statements in the programs they produce. More recently I discovered why the use of the go to statement has such disastrous effects, and I became convinced that the go to statement should be abolished from all “higher level” programming languages (i.e. everything except, perhaps, plain machine code).⁶

Criticism about the undisciplined use of go to statements was one of the factors that led to the *structured programming* movement of the 1970s. Structured programming aims to improve the clarity, quality, and development time of software by identifying coherent blocks of code and using *subroutines*, standard sets of instructions to perform frequently used operations within a program. Structured programming uses *for loops* and *if-then-else* constructs as the only control and decision structures.

Recursion and dynamic data structures

Since the beginning of computer programming, most programming languages have been imperative languages that tell the machine how to do something. It is remarkable, however, that one of the earliest high-level programming languages to be developed was a declarative language, which told the computer what the programmer would like to have happen and let the machine determine how to do it. This declarative language was LISP, the brainchild of John McCarthy, who had spent the summer of 1958 as a visiting researcher in the IBM Information Research Department. In addition to becoming frustrated with batch processing and wanting to pursue time-sharing systems, he investigated the requirements for a programming language to support symbolic rather than purely numeric computations. These requirements included *recursion*, the ability



B.4.2. Edsger Dijkstra (1930–2002) was one of the pioneers of computer science. From the early days he was championing a mathematically rigorous approach to programming. In 1972 he received the Turing Award for his fundamental contributions to the development of programming languages. He was well known for his forthright opinions on programming and programming languages.

On FORTRAN:

FORTRAN, “the infantile disorder”, by now nearly 20 years old, is hopelessly inadequate for whatever computer application you have in mind today: it is now too clumsy, too risky, and too expensive to use.³¹

On COBOL:

The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offense.³²

On BASIC:

It is practically impossible to teach good programming to students that have had a prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration.³³

for a program or subroutine to *call* itself, and *dynamic data structures*, for which the memory space required is not set in advance but can change size as the program runs. The newly developed FORTRAN language did not support either of these requirements.

When he returned to MIT, McCarthy set about devising a language suitable for the types of symbolic and AI applications he had in mind. His LISP language focused on the manipulation of *dynamic lists* that could grow or shrink as the program was running. The style of programming was very different from that of an imperative language like FORTRAN. There were no assignment statements in LISP and no implicit model of state in the background, tied to the physical implementation of a “pigeonhole” model of computer memory. Instead, in LISP everything is a mathematical function. For example, the expression x^2 is a *function*: applying this function to the variable x returns the value of x^2 . In LISP, computation is achieved by just applying functions to arguments. This feature makes it easier to reason mathematically about the behavior and correctness of LISP programs. It removes the possibility of dangerous “side effects” – situations in an imperative program where, unsuspected by the programmer, some part of the program has altered the value of a variable in memory from the value that the programmer had intended. Memory allocation for McCarthy’s dynamic lists was not performed in advance of running the program. It was therefore necessary to regularly clean up the memory space by removing from the list of assigned storage locations any locations that were no longer being used. McCarthy’s team called this process of reclaiming memory space *garbage collection*. These ideas have been influential in the implementation of modern languages like Java and C# (C Sharp).

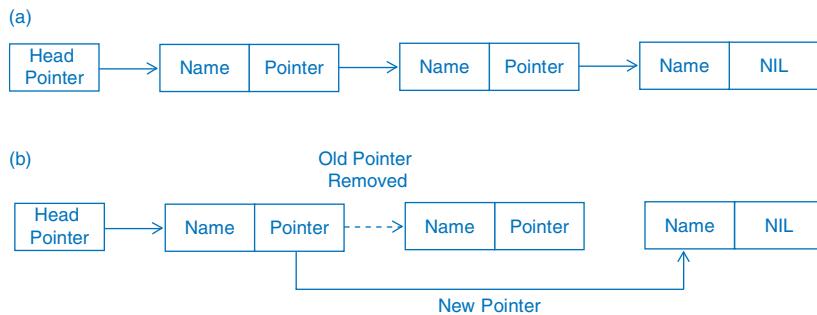
McCarthy’s ideas about recursion and dynamic data structures are not limited to declarative languages like LISP. They are now essential components of almost all imperative languages, such as FORTRAN, BASIC, and C. We can illustrate the idea of recursion through the problem of calculating the factorial of the number n – that is, the product of the whole number n and all the whole numbers below it. The factorial of n is written as $n!$ and is defined to be the product $n \times (n - 1) \times (n - 2) \dots \times 2 \times 1$. We could calculate this using a *for* loop, but we can also perform the calculation using recursion, a process in which the function repeatedly calls itself with smaller and smaller arguments until it reaches a *termination condition*, a state in which it will stop calling itself. The pseudocode for the recursive calculation of factorial n is:

```
factorial (n)
if n <= 1:
    return 1
else:
    return n * factorial (n - 1)
```

The expression “ $n \leq 1$ ” is an integer check to see if the value of n is less than ($<$) or equal ($=$) to 1. The calculation of the factorial starts with n and multiplies this by factorial ($n - 1$). This repetition continues until ($n - 1$) equals 1.

As an example of dynamic data structures we consider *linked lists*. In scientific calculations, the size of the array structures that store related groups of

Fig. 4.5. A linked list is an example of a dynamic data structure that makes it easy to remove or add items to a list. (a) This shows the structure of a linked list with each entry in the list having a pointer to the memory location of the next element. (b) This illustrates how easy it is to delete a node from the linked list by just changing the pointer.



items can usually be specified in advance. The same is not true for most types of lists. The membership list for a sports club, for example, will grow and shrink as new members join and old members leave. If we store a list of names in an array of fixed length using sequential memory locations in the computer, removing or adding names becomes very laborious because the data in the list must be frequently reshuffled to keep them in the right order in memory. These problems can be avoided if we do not store items in the list in a sequential memory block but just in any convenient area of memory. To store the contents of the list, each name in the list is stored in some location along with a *pointer* – a memory address – to the location of the next name on the list (Fig. 4.5a). Deleting or adding names is now straightforward because it just requires changing a single pointer (Fig. 4.5b). With pointers, the address of the data storage location can be kept separately from the actual data. Retrieving the data is thus a two-step process – getting the address of the storage location and then going to that location to get the data. Other common dynamic data structures are LIFO (Last-In-First-Out) stacks, collections of items in which only the most recently added item may be removed, and FIFO (First-In-First-Out) queues, collections of items in which the earliest added item may be removed.

Programming with objects: from SIMULA to C++

An important idea in object-oriented programming is *data abstraction*, which focuses on classes, objects, and types of data in terms of how they function and how they can be manipulated, while hiding details of how the work is carried out. The idea of data abstraction can be traced back to two Norwegian computer scientists, Kristen Nygaard and Ole-Johan Dahl (B.4.3). They first presented their programming language SIMULA 67 in March 1967. They were interested in using computers to run simulations and needed the language to support subprograms that could stop and later restart at the place they had stopped. To do this, Nygaard and Dahl introduced the idea of a *class*. The key property of a class is that a data structure and the routines that manipulate that data structure are packaged together. This led to the important idea of *abstract data types*, sets of objects that share a common structure and behavior.

Abstract data types were actually present in the original FORTRAN language. There was a built-in *floating-point* data type, which represented



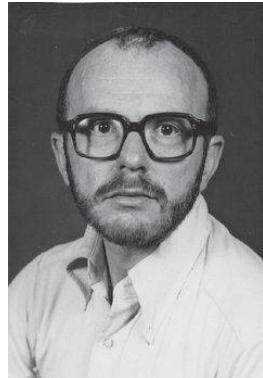
B.4.3. Ole-Johan Dahl (1931–2002) (left) and Kristen Nygaard (1926–2002) were first to introduce classes and objects in their Simula programming language.

numeric values with fractional parts, and a set of arithmetic operations that were allowed to act on floating-point variables. How FORTRAN included these data types is an example of one of the key ideas of data abstraction, namely, *information hiding*. The details of the actual way a floating-point number is represented in the computer are hidden from the user and cannot be accessed by the programmer. In addition, the programmer can only create new operations on this type of data by using the built-in operations already supported on floating-point variables. Because of such information hiding, FORTRAN programs could run on many different machines, even though floating-point variables were frequently implemented very differently on different machines.

Modern object-oriented (O-O) programming languages allow programmers to create their own abstract data types. Consider again the problem of writing a program to manipulate a list containing the names of members in a sports club. In an imperative programming language, the list is just a collection of data and we need to write separate software procedures to add, delete, and sort items in the list. In an O-O language, the list is constructed as an *object* consisting of both the list data and the collection of procedures – called *methods* in O-O speak – for manipulating this data. Thus an O-O program to sort the list would not contain a separate sorting procedure but make use of the methods already built in for the list object. What is the relationship between a class and an object? A *class* is a template for all the objects with the same data type and methods. The list class applies to all list objects with data in the form of a list and the methods to operate on the list. As a slightly more complicated example, let us define a bank account class. The abstract data type for a bank account consists of the name of the client, the number of the account, and the balance of money in the account. The class consists of bank account data of this type plus methods that define the different operations that can be carried out on the account – withdrawals, deposits, transfers, and so on. Accounts belonging to different customers obviously contain different data and are called *objects* or *instances* of this class. The methods that act on the data within an object are usually small imperative programs.

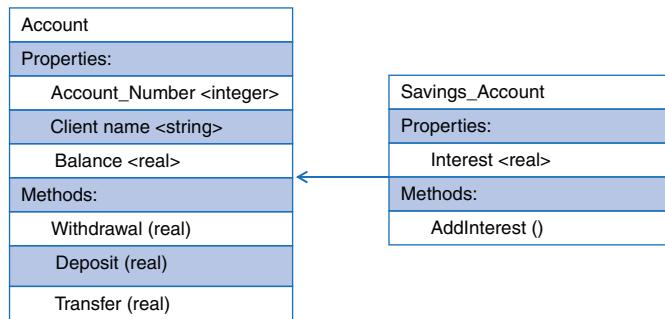
Two other important properties of O-O languages are *inheritance* and *encapsulation*. The idea of inheritance is that a class can be extended to create another class that inherits the properties of the original class. Thus the class “bank account” could be extended to create a new class “savings bank account” that inherits the same data structure and methods as the original class but with additional properties and methods (Fig. 4.6). Encapsulation means that there are certain properties of an object that are not accessible to other parts of the program. Only the object is able to access these properties.

Canadian computer scientist David Parnas (B.4.4) was one of the pioneers of information hiding. Turing Award recipient Alan Kay and his research team at the Xerox PARC (Palo Alto Research Center) in Silicon Valley in the 1970s first introduced the term *object-oriented programming*. They developed the Smalltalk language, which was based on the idea of building programs with objects that communicated by sending messages.



B.4.4. David Parnas is a Canadian computer scientist who pioneered ideas of “information hiding.” These ideas are now an integral part of data abstraction in object-oriented programming.

Fig. 4.6. This figure illustrates the concept of class inheritance. The properties of the Account class are the data items. The methods represent the actions that we can carry out on the data. For the Savings_Account class, in addition to the properties and methods inherited from the Account class, there is also a new data item called interest and a new AddInterest method. In this way we can construct more complex classes from simpler ones.



One of the most widely used O-O languages today is the C++ programming language. Bell Labs researcher Bjarne Stroustrup (B.4.5) was familiar with SIMULA and had found the class feature to be useful in large software development projects. When he started working at Bell Labs, he explored ways of enhancing Dennis Ritchie's C language, which was both fast and portable (see section on C and Unix at the end of this chapter). In 1979, Stroustrup started by adding classes to C to create what he called "C with Classes." Over the next few years, Stroustrup added several other features and renamed the language "C++." There is now a C++ software library called the Standard Template Library (STL) that contains predefined, useful classes that are provided as part of the C++ programming environment. By incorporating the STL library of classes into a program, the programmer does not have to explicitly specify these data structures. Two other examples of widely used object-oriented programming languages are Java (B.4.6) and C#.



B.4.5. Bjarne Stroustrup designed and implemented the C++ programming language. Over the last two decades, C++ has become the most widely used language supporting object-oriented programming and has made abstraction techniques affordable and manageable for mainstream projects.

Why do we need software engineering?

As we have seen, computer scientists originally hoped that programming in a high-level language would, as Backus said, "virtually eliminate coding and debugging."⁷ For small scientific programs written by one or two researchers, programming certainly became much easier, with the hard work of converting a FORTRAN program into efficient assembly code delegated to a computer program, the compiler. However, many scientific programs nowadays are complex simulation codes incorporating many different aspects of the problem under investigation. Writing and debugging such programs has become much more



B.4.6. James Gosling is credited with the development of Java programming language. The name can be traced back to the brand of coffee fueling the programming effort. A distinguishing feature of a Java program is that it does not run directly on the hardware but on software called a "virtual machine." This "architecture-independent" implementation enables that movement of the code from one computer to another without recompiling the code. Thanks to this "write once, run anywhere" principle Java has become one the most popular programming languages especially for web applications.

difficult. An additional challenge is that new researchers, who were not the authors of the original program, may need to extend and modify the code.

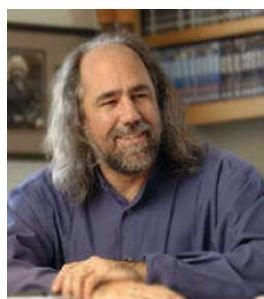
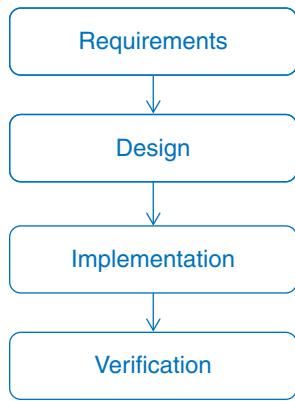
Similarly, as the software for business applications became larger and more complicated, it was no longer possible for a small team of talented programmers to write all the code. Teams of hundreds or even thousands of programmers now work on software systems consisting of hundreds of thousands or millions of lines of code, and programmers now have to coordinate their work. Accurate estimates of the time and cost of writing a complex software system have become vitally important for software companies. Brooks discusses these issues in his book on the “mythical” man-month unit of programming effort. As a result of his experience, he formulated “Brooks Law,” which says, “Adding manpower to a late software project makes it later.”⁸

Software companies have an urgent need for reliable answers to the questions: How many lines of code will it take to provide the desired functionality? How many programmers will be needed? How long will it take? If a software project is behind schedule, what should you do? Software engineering attempts to define methodologies and frameworks to answer these questions. The Institute of Electrical and Electronics Engineers (IEEE), a professional organization devoted to promoting technological innovation, defines software engineering in its Standard 610.12 as “The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.”⁹

One of the earliest attempts to apply engineering methodologies to software development was the “waterfall” model (Fig. 4.7). This identified four distinct phases of software development – requirements analysis, design, implementation, and testing. The waterfall method calls for completing each phase before proceeding to the next, which requires the systematic description and documentation of both the requirements and the design of the software to be completed before any actual coding begins. In practice, the phases are rarely completely separate. Software developers often find in later phases that they must go back and change things in earlier phases. Backtracking and multiple versions of each phase are common. David Parnas says about the design process:

Even if we knew the requirements, there are many other facts that we need to know to design the software. Many of the details only become known to us as we progress in the implementation. Some of the things we learn invalidate our design and we must backtrack.¹⁰

The recognition that software development is not a linear process has led to a philosophy called *agile software development*. Agile methods break the task of writing the whole system into smaller segments or “sprints,” each of which involves all the four phases of software development – analyzing requirements, designing, implementing, and testing. Based on the results of testing the latest version of a design, the developers make changes and improvements. The sprints typically last around four weeks, and the goal is to have a working prototype with some of the functionality required of the final product by the end of each sprint. One of the main motivations for these more flexible software engineering methodologies is that customers often do not know all their



B.4.7. Grady Booch is an evangelist of the systematic approach to software design. In one of the interviews he referred to his mission with the following words: "if I had not discovered software I would have been a musician or a priest." He is one of the authors of the UML, which represents a framework for constructing and reasoning about the software. UML is a collection of diagrams and tools that allows programmers to cope with complex systems by raising the level of abstraction.

Fig. 4.7. The waterfall model is one of the earliest methods used to systematize software development. In principle, the model consists of several independent stages with each stage feeding to a subsequent stage. This approach is sometimes referred to as "Big Design Up Front" – because a new stage can only start if the preceding one has been fully completed. This is the strength of the method – but also its weakness. In reality, software development is not a linear process and many issues cannot be foreseen until later stages in the project. So, in practice, the individual stages are not fully isolated from each other: often we need to backtrack to make revisions and changes in the previous stages.

requirements at the beginning of a project. To incorporate changes in requirements makes a less rigid approach than the formal approach of the waterfall model essential.

The first stage of the software life cycle is requirements analysis and specification. One of the earliest tools for documenting computer programs was the *flowchart*, a diagram representing the sequence of operations in a program. We have already seen examples of flowcharts for *for* loops and *if-then-else* control statements (see Fig. 4.4). However, in the production of large, complex software systems, flowcharts have proved to be of limited value. In the 1980s, computer scientist David Harel was working with avionics engineers trying to specify the behavior of a software system to control a modern jet aircraft. An avionics system is *reactive* – a term coined by Harel and his colleague the late Amir Pnueli – in the sense that it has to respond predictably to a wide variety of different types of events. Harel eventually converged on a diagrammatic way to specify the responses and transitions of the avionics system, which he called *statecharts* – "the only unused combination of 'state' or 'flow' with 'chart' or 'diagram.'"¹¹ By 1986, Harel and his colleagues had built the *Statemate* tool, which not only allowed users to construct statecharts but was also able to automatically generate code to fully execute them. In the 1990s, they developed an O-O version of statecharts, which later became the heart of the Unified Modeling Language, or UML. UML was devised in 1996 by Grady Booch (B.4.7), James Rumbaugh, and Ivar Jacobson. It is a collection of visual languages for specifying, constructing, and documenting complex software designs. Booch comments, "If you look across the whole history of software engineering, it's one of trying to mitigate complexity by increasing levels of abstraction."¹² The UML approach (Fig. 4.8) is yet one more attempt to reduce the complexity of software production.

The final phase of the software life cycle is testing and maintenance. For complex software systems, it is impossible to test all branches of the code under all possible combinations of input data and initial states. According to Dijkstra, "Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence."¹³ A 2002 report from the National Institute of Standards and Technology, a U.S. government agency that works to promote innovation and industrial competitiveness, estimated that inadequate software testing cost the U.S. economy nearly \$60 billion per year. The report also stated, "In fact, the process of identifying and correcting defects during the software development process represents approximately 80 percent of development costs."¹⁴

Testing a modern software system involves the application of a variety of different tools. *Dynamic software testing* involves running the code using a set of

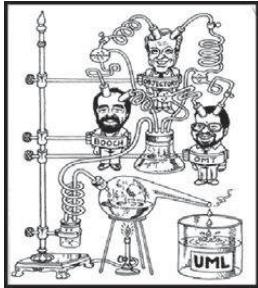


Fig. 4.8. UML is a general methodology that allows a systematic “step-by-step” approach to software design. UML is a unification of three software techniques developed by Grady Booch (Booch), Ivar Jacobson (Objectory), and James Rumbaugh (OMT); although UML also has roots in David Harel’s statecharts. In software engineering circles Booch, Jacobson, and Rumbaugh are often referred to as the “three amigos.” UML diagrams allow evaluation of various implementation options prior to actual program coding.

test cases. *White-box testing* is designed to test the internal structures of a program. The tester attempts to choose sets of inputs that exercise all the different possible paths through the code. *Black-box testing* takes the view of the user rather than the software developer. The tester checks the software’s functionality with no knowledge of the system’s internal structure. Another important type of evaluation is *fuzz testing*, in which valid input data sets are modified with random mutations and then fed into the program. Providing such invalid and unexpected inputs to the system allows the tester to determine how a program handles *exceptions* – unpredictable conditions or situations that can cause a program to crash or possibly create a security risk in the software (Fig. 4.9).

Empirical software engineering

The increasing complexity of modern software development is indicated by the numbers of programmers and lines of code in three releases of the Microsoft Windows operating system (Fig. 4.10). Here the programmers are divided into “developers,” who write the code, and “testers,” who systematically check the code for bugs. To allow such large numbers of programmers to work on different parts of the software system simultaneously, Microsoft developed a *synchronize-and-stabilize* approach to writing software. Breaking up the software into several different “branches” that can be worked on at the same time allows “large teams to work like small teams.” Much of the complexity now lies in the process of correctly joining the branches back together. Microsoft solved the problem using “daily synchronizations through product builds, periodic milestone stabilizations, and continual testing.”¹⁵ Microsoft also developed an error-reporting tool so that users could inform the company of any software problems. Analysis of the data led to some interesting conclusions, as summarized by former Microsoft CEO Steve Ballmer:

One really exciting thing we learned is how, among all the software bugs involved in reports, a relatively small proportion causes most of the errors. About 20 percent of the bugs cause 80 percent of all errors, and – this is stunning to me – one percent of bugs cause half of all errors.¹⁶

Fig. 4.9. A screenshot of the dreaded moment when a computer crashes. In programmer circles such an event is known as “the blue screen of death.”

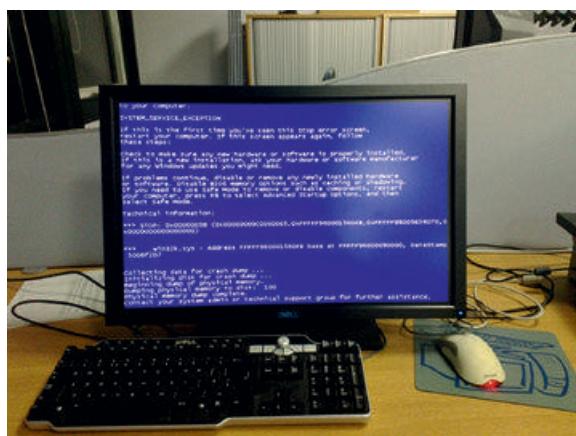


Fig. 4.10. Size and scale of the programming teams and code base for versions of the Microsoft Windows operating system.

Ship Date	Product	Development Team Size	Test Team Size	Lines of Code (LOC)
July 1993	Windows NT	200	140	5 million
December 1999	Windows 2000	1,400	1,700	30 million
October 2001	Windows XP	1,800	2,200	40 million

In his book *Code Complete*, Steve McConnell estimates the extent of the bug problem:

Industry average experience is about 1–25 errors per 1000 lines of code for delivered software. The Applications Division at Microsoft experiences about 10– 20 defects per 1000 lines of code during in-house testing and 0.5 defects per 1000 lines of code in released product.¹⁷

The problem of deciding which bugs to fix and which are likely to generate new errors is complex. This is an area where the new field of empirical software engineering aims to help. The *Journal of Empirical Software Engineering* says:

Over the last decade, it has become clear that empirical studies are a fundamental component of software engineering research and practice: Software development practices and technologies must be investigated by empirical means in order to be understood, evaluated, and deployed in proper contexts. This stems from the observation that higher software quality and productivity have more chances to be achieved if well-understood, tested practices and technologies are introduced in software development. Empirical studies usually involve the collection and analysis of data and experience that can be used to characterize, evaluate and reveal relationships between software development deliverables, practices, and technologies.¹⁸

This statement has now been adopted as part of the manifesto of the International Software Engineering Research Network.

One example of this empirical approach to software engineering is the CRANE tool developed by researchers at Microsoft – where CRANE is an acronym formed from Change Risk ANalysis and impact Estimation. The CRANE project looked at the challenges of providing support for multiple versions of Windows, running on a wide variety of computers, with a user base of more than a billion. One immediate challenge is that software maintenance for a released product is done by different teams of software engineers than those who developed the software. The goal of the CRANE project was to use historical information about the software being serviced to build risk-prediction models using advanced statistical techniques that could guide bug fixing and testing. For every bug in any software component, the tool provides the following information: what has happened to the component so far in servicing; what exactly is being changed with the proposed fix; which fixes carry more than average risk of causing more bugs; which tests to run after the change; which other components to test in addition to the changed component; and

which applications are potentially impacted by the change. Such empirical software engineering tools are enabling maintenance software engineers to make informed, data-driven decisions about their priorities.

Open-source software

A very different model of software development is the philosophy promoted by the *open-source software* movement. One of the origins of this movement was the decision of AT&T to allow the distribution of the Unix source code under a “free” license (see section on Unix and C). Bell Labs researchers Ken Thompson and Dennis Ritchie wrote the Unix operating system in the early 1970s. It was the first operating system to be written in a high-level language, the C programming language developed by Ritchie. The source code for the C compiler developed by Bell Labs researcher Stephen Johnson was also freely distributed with the Unix code. For only a few-hundred-dollar licensing fee, the university research community could obtain not only a functional operating system but also a platform for teaching and research. In 1956, AT&T had settled an antitrust monopoly suit with the U.S. Department of Justice, and AT&T’s lawyers interpreted the agreement as forbidding the company to enter new markets not related to telephones. The AT&T license agreement for Unix was intended to make it crystal clear that the company was not creating a new business with computers:

The terms of the early Unix licenses were minimal: The software came “as is” with no royalties to AT&T, but also no support and no bug fixes.¹⁹

One immediate result of this license agreement was to encourage the research community to set up self-help networks and share information on bug fixes. This set the style for the development of a global Unix support and development community with developers freely sharing their suggested code changes. The most significant research collaboration focused on Unix was between the original Bell Labs team of Richie and Thompson and the Computer Systems Research Group (CSRG) at the University of California, Berkeley. In 1983, the CSRG team released the latest version of their “Berkeley Unix” software, known as 4.2 BSD. This software incorporated the new Internet protocols and allowed Unix systems to be easily connected to the rapidly growing Internet. The initials BSD stand for Berkeley Software Distribution, which included an open-source software license

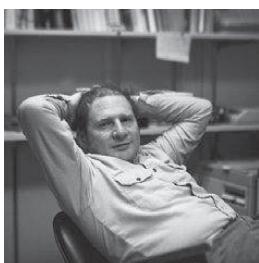


B.4.8. Richard Stallman is the originator of the free software movement. In 1979 he was working in the AI lab at MIT when the lab installed a new laser printer from Xerox. The printer suffered from paper jams and Stallman wanted access to the source code of the printer driver so he could modify it and fix the problem. Xerox would not give him the source code and he ended up being very frustrated. In 1984 Stallman resigned from MIT to set up the Free Software Foundation. Stallman was very explicit in his explanation of “free”: “Since free refers to freedom, not to price, there is no contradiction between selling copies and free software.”²⁴ He called his project to build a free operating system by the recursive acronym GNU – standing for GNU’s Not Unix. He also devised the GPL source license that was designed to ensure that any modifications to the source code were covered by the same license, including combinations of GPL software with commercial software.

that allowed free use of the source code. Importantly, the license allowed the possibility of incorporating all or part of the Berkeley software in a closed-source commercial product. The license only required that any copyright notices in the code were maintained along with the disclaimer of any warranty.

The 1980s were a confusing time for the Unix community. By this time, AT&T had realized that Unix was a very valuable software product and, under the terms of a new antitrust settlement in 1984, the company began charging for the Unix software. By 1992, friction between AT&T's new, commercially focused Unix Systems Laboratories division and the freewheeling Berkeley open-source community had come to a head. AT&T began a court case against the University of California. In addition to these legal problems, many different and incompatible variants of Unix had been spun-off – “forked” – from the original open-source Unix code, leading to a very fragmented Unix development community. Meanwhile, at MIT, a software developer in the AI lab named Richard Stallman (B.4.8) had become concerned about the loss of community that happened when software could not be freely shared. In 1984, Stallman founded the Free Software Foundation (B.4.9) with the goal of developing “an entirely free operating system that anyone could download, use, modify, and distribute freely.”²⁰ He named his project GNU, standing for “GNU’s Not Unix.” To ensure that the source code remained open and freely shareable, Stallman devised the GNU Public License (GPL) that is very different from the permissive Berkeley BSD open-source license. The GPL license requires that any modifications of the software must be released under the same GPL open-source license. More important for commercial software companies was the “viral” requirement that any software formed by combining free, GPL-licensed software with commercial software must all be released under a free GPL license. Under the GNU umbrella, Stallman created some very popular tools for writing software that are still widely used by the computer science community – the *GNU Emacs* text editor, the *GCC* compiler, and the *GDB* debugger. However, it was left to a young Finnish graduate student named Linus Torvalds (B.4.10) to reunite the Unix community around his version of the Unix kernel, the core component of the Unix operating system.

In 1991, Torvalds was a graduate student at the University of Helsinki and had bought himself a new personal computer (PC) based on Intel’s 386 microprocessor. Because he wanted to run Unix on his PC, he bought and installed Minix, a version of Unix suitable for teaching that had been created by Andy Tanenbaum at the Vrije Universiteit in Amsterdam. Inspired by the Minix software, Torvalds started creating his own version of the Unix kernel for the PC.



B.4.9. Hal Abelson is a professor of electrical engineering and computer science at MIT. He is passionate about both open-source software and open courseware, and has been a champion for the right to open access for publicly funded research publications. Abelson was one of the founders of the Free Software Foundation and the Creative Commons movements. In addition, Abelson has long believed in the potential for using computation as a conceptual framework in teaching. He is the author of several influential textbooks and implemented the Logo programming language on Apple II computers. Logo is widely regarded as one of the best programming languages for introducing computing to children. His pioneering work in education was recognized in 2012 by his receiving the ACM SIGCSE Award for Outstanding Contributions to Computer Science Education.



Fig. 4.11. Linux celebrates twenty years with release 3.0. Tux is the official mascot of the Linux community. According to legend, Torvalds was looking for something fun and sympathetic to associate with Linux, and a slightly fat penguin sitting down after having had a great meal perfectly fit the bill.

In 1991, he made the source code of his new operating system, called Linux (Fig. 4.11), available on the Internet with the following announcement:

I'm working on a free version of a Minix look-alike for AT-386 computers. It has finally reached the stage where it's even usable (though it may not be, depending on what you want), and I am willing to put out the sources for wider distribution.... This is a program for hackers by a hacker. I've enjoyed doing it, and somebody might enjoy looking at it and even modifying it for their own needs. It is still small enough to understand, use and modify, and I'm looking forward to any comments you might have. I'm also interested in hearing from anybody who has written any of the utilities/library functions for Minix. If your efforts are freely distributable (under copyright or even public domain) I'd like to hear from you so I can add them to the system.²¹

Torvalds was surprised by the response to his invitation from the worldwide Unix community. Within a couple of years, hundreds of developers had joined his Internet newsgroup and were contributing bug fixes, improvements, and new features to Linux. By 1994, Torvalds was able to release the first complete version of his operating system, Linux version 1.0. This listed nearly eighty developers as contributors, from a dozen different countries. From these modest beginnings, Linux has become much more than a hobbyist's PC operating system. By 1999, Red Hat and VA Linux were established as public companies offering "Linux support" – although the basic code was still freely available. By 2000, Linux had received official recognition from IBM, which announced it would offer enterprise support for Red Hat Linux on their mainframe computers. Major software companies such as Oracle Corporation and SAP soon followed, and by 2013 Linux had become established as a major component of both university and business software environments.

Who are the developers who contribute to Linux? One recent study found that there were more software developers from industry than from universities and research organizations. It is also probably true that, over the last decade or so, several hundred professional software engineers from companies like IBM and Intel have participated in major open-source projects. Another survey found that 10 percent of the developers are credited on more than 70 percent of the code. In his book *The Success of Open Source*, Steven Weber concludes:



B.4.10. Linus Torvalds is credited with the development and maintenance of the Linux kernel, which has become the basis for most popular open-source operating systems. In the programmer community he is considered as a "benevolent dictator" who makes sure that the released code is always in perfect shape. Despite the fact that it took him eight years to get his master's degree at the University of Helsinki, he turned out to be a very successful programmer. He described the development of the Linux kernel in a book *Just for Fun*. Most of his concern is not the technical side of Linux but the software patents that are notoriously difficult to deal with.

These numbers count only the major contributors to the Linux kernel. Other active developers report and patch bugs, write small utilities and other applications, and contribute in less elaborate but still important ways to the project. The credit for these kinds of contributions is given in change logs and source code comments, far too many to read and count in a serious way. It is a reasonable guess that there are at least several thousand, and probably in the tens of thousands, of developers who make these smaller contributions to Linux.²²

How is the work of these volunteer contributors organized? Unlike the formal software engineering frameworks described earlier, with open-source software development there is no authority other than consensus. In the case of Linux, Torvalds still acts as a sort of benevolent dictator supported by a small number of key lieutenants. Other open-source efforts have a small core team who make the decisions about what code to accept. This informal model of software development has produced a complex modern operating system consisting of millions of lines of code with a quality and stability that can rival that of commercial software.

There are now thousands of open-source software projects addressing a large number of different application areas. For many university computer science departments, the use of open-source software for research is the standard way of working. In 2013, SourceForge, a popular site for open-source software projects, stated, “3.4 million developers create powerful software in over 324,000 projects,”²³ which works out as an average of about ten developers per project. In addition, the SourceForge directory “connects more than 46 million consumers with these open source projects and serves more than 4,000,000 downloads a day.”²⁴ Even though only a very small number of these projects attract a critical mass of developers and attain widespread use, the open-source software development model has clearly proved to be a viable alternative to traditional software development methodologies.

Scripting languages

Another type of programming language that is increasing in popularity is a group of languages known as *scripting languages*, high-level programming languages that are interpreted by another program at runtime rather than needing a compiler to transform the source code into an executable program. A *shell script* in Unix was a sequence of commands that could be read from a file and executed in sequence, as if they had been typed in using a keyboard. By extension, the term *script* has become used to describe a set of instructions executed directly by the computer rather than needing a compiler like a traditional programming language. Today, scripting languages have become much more powerful than these early examples because of the addition of standard programming language concepts, such as loops and branches. There are two main uses for scripting languages. The first is as a “glue” language that allows applications to connect off-the-shelf software components that are written in a conventional programming language. The second exploits the functionality and ease of use of scripting languages to employ them as an alternative to conventional languages for a range of general programming tasks.

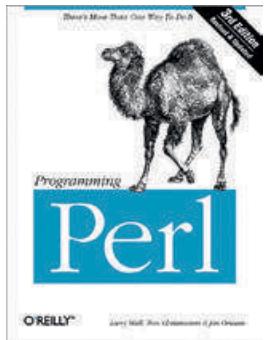


Fig. 4.12. A guide to programming in Perl by its creator, Larry Wall, with Tom Christiansen, and Jon Orwant, widely known as the “Camel” book.

A major characteristic of modern scripting languages is their interactivity, sometimes referred to as a *REPL* programming environment. *REPL* stands for “Read-Eval-Print Loop” and has its origins in the early work on LISP at MIT. When a user enters an expression, it is immediately evaluated and displayed. In this sense, scripting languages behave as if they were “interpreted,” meaning that they operate on an immediate line-by-line execution basis. This is in contrast to traditional “compiled” languages, which generate a binary object file that needs to be explicitly linked to the required set of program libraries – the traditional “edit-compile-link-run” cycle of programming. Because of the increasing power of today’s computer chips, the ease of use of scripting languages is often more important than the increase in program efficiency that can be achieved with a compiled language. For example, in scripting languages, to minimize the complexity of programs, declaration of variable types to designate the sort of information each variable can contain is often optional. The variable types are declared implicitly by their usage context and initialized to be something sensible when first used. However, as scripting language programs have become longer and more complex, the benefits of type declarations have been recognized, and most scripting languages now provide an option to make explicit type declarations.

The development of Perl (Practical Extraction and Report Language) in the late 1980s (B.4.11) was one of the defining events in the evolution of scripting languages (Fig. 4.12). David Barron in his book *The World of Scripting Languages* remarks that:

Perl rapidly developed from being a fairly simple text-processing language to a fully-featured language with extensive capabilities for interacting with the system to manipulate files and processes, establish network connections and other similar system-programming tasks.²⁵

From its origins in the Unix world, Perl scripts are now able to run unchanged on all the popular operating system platforms. Other popular scripting languages are VBScript (Visual Basic Scripting Edition) for the Microsoft platform and JavaScript for web applications. The characteristics of ease of use and immediate execution with a *REPL* environment are sometimes taken as the definition of a scripting language. Under this definition, the Python programming language, which is growing rapidly in popularity, would be regarded as a scripting language.

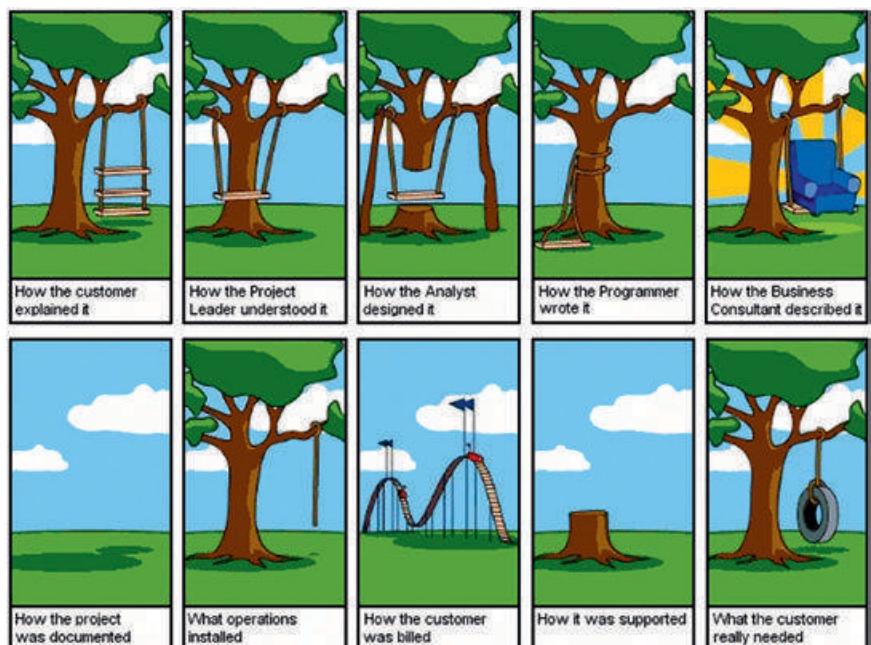


B.4.11. Larry Wall developed Perl in 1987 as a general-purpose Unix scripting language. Since then both the portability and the features in Perl have been expanded greatly and now include support for O-O programming. It is one of the world’s most popular programming languages. Wall continues to oversee the evolution of Perl and his role is summarized by the so-called 2 Rules, taken from the official Perl documentation:

1. Larry is always by definition right about how Perl should behave. This means he has final veto power on the core functionality.
2. Larry is allowed to change his mind about any matter at a later date, regardless of whether he previously invoked Rule 1.

Key concepts

- Strong typing
- Control structures – for loops and if-then-else
- Recursion
- Dynamic data structures – linked lists, stacks, and queues
- Data abstraction and information hiding
- Object-oriented programming
 - Classes and objects
 - Inheritance and encapsulation
- Software life cycle
 - Requirements analysis
 - Design
 - Implementation
 - Testing
- Waterfall method and agile methods for software engineering
- Empirical software engineering
- Formal methods
- Open-source software development
- Scripting languages



Some more background on software topics

Unix and C

The origins of time-sharing operating systems can be traced back to MIT, with John McCarthy's early prototype and Fernando Corbató's Compatible Time-Sharing System in 1961. These beginnings led to Licklider's very ambitious project MAC and the Multiplexed Information and Computing Service – Multics – time-sharing operating system. Bell Labs were partners in the project but became frustrated by its size, complexity, and slow progress. When Bell Labs withdrew from the project in 1969, Thompson and Ritchie (B.4.12) and some colleagues from Bell Labs decided to produce their own stripped down version of a time-sharing operating system and to try to create a community around the new code base:

What we wanted to preserve was not just a good environment in which to do programming, but a system around which a fellowship could form. We knew from experience that the essence of communal computing, as supplied by remote-access, time-shared machines, is not just to type programs into a terminal instead of a key-punch, but to encourage close communication.²⁶

They were unable to get funding from Bell Labs management to buy a new computer for their project so they found an old and little-used PDP-7 minicomputer to begin their entirely unfunded *skunkworks* project. In the course of this work they developed a hierarchical file system, the concept of treating devices as files and the notion of processes. They also created a set of utilities giving users the ability to print, copy, delete, and edit files plus a simple command interpreter or *shell*. The Unix operating system then consisted of a set of utilities under the control of a small and efficient operating system *kernel*. The kernel provided services to start and stop programs, handle the file system, and schedule access to resources and devices avoiding conflicts. By promising to create a system specifically designed for editing and formatting text, in 1970 they finally managed to get funding to buy a modern PDP-11 computer. It was also in 1970 that their colleague Brian Kernighan suggested the name Unix, as a play on the name Multics. In 1972 Unix pipes were introduced that enabled small utility programs to be combined to create more powerful programs. Using such pipes to create a powerful system utility rather than developing a single monolithic program with the same combined functionality became known as the Unix philosophy – “the idea that the power of a system comes more from the relationships among programs than from the programs themselves.”²⁷ Every program in Unix had originally been written in assembly language but Thompson had developed a definition and compiler for a new language for the PDP-7 that he called B. The language was a stripped down and modified version of the BCPL language developed in Cambridge, U.K., by Martin Richards. With the arrival of the more powerful PDP-11 in 1970, Ritchie took Thompson's B

language and developed the C programming language to take advantage of the new machine's byte addressability and other features. By 1973 the new language was powerful enough for much of the Unix kernel to be rewritten in C. In 1977, with further changes to the language, Ritchie and Steven Johnson were able to produce a portable version of the Unix operating system. Johnson's Portable C Compiler then allowed both C and Unix to spread to other platforms. In 1978 Kernighan and Ritchie published the first edition of *The C Programming Language*, which served for many years as the informal specification of the standard (Fig. 4.13).

Thompson and Ritchie's Unix operating system has been enormously influential. Because AT&T was a regulated telephone monopoly it was barred from doing significant commercial developments in



B.4.12. Ken Thompson and Dennis Ritchie (1941–2011), the developers of C programming language and the Unix operating system.

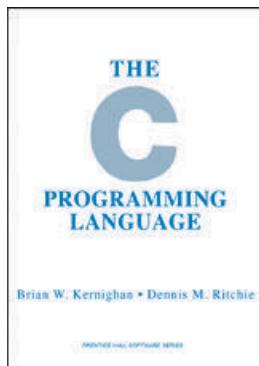


Fig. 4.13. The first edition of Kernighan and Ritchie's C programming language book.

the computing arena. Universities were therefore able to order the data tapes from Bell Labs for a nominal charge of \$150 for materials. For that they received the entire source code for the first general-purpose operating system for minicomputers. This along with the portable C compiler was all that university researchers needed to produce their own versions. After July 1974, when a written version of their work appeared in *Communications of the ACM*, orders came flooding in and first hundreds, and then thousands of minicomputer users started porting Unix to their machines. Thompson spent a sabbatical year in Berkeley in 1975 and a graduate student named Bill Joy became an enthusiastic promoter of Unix. By the early 1980s, the Berkeley System Distribution 4.2 Unix was the de facto standard in the university research community. Joy and his team were then commissioned by ARPA, the Advanced Research Projects Agency, to integrate the newly defined networking protocol TCP/IP into Unix. This was a very significant development for the birth of the Internet as we shall see in Chapter 10.

Formal methods

Software engineering involves many disciplines, including mathematics. In the context of software development the field of *formal methods* uses a variety of mathematical techniques to specify and verify software (B.4.13). A formal specification of the system can be used to prove that the program has the desired properties. Automated *theorem proving* systems attempt to prove that the software does what it was intended to do by using its formal specification, a set of logical axioms and a set of inference rules to produce a formal mathematical proof. An alternative approach uses *model checking*, which verifies properties of the system by an exhaustive search of all the possible states that the system could enter during its execution. It is probably fair to say that formal methods have not so far delivered major benefits for assisting the creation of bug-free code in large software systems. However, there are now some examples of such methods being used to solve real software problems. In 2002 Bill Gates said:

Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability.²⁸



B.4.13. Three pioneers of structured programming and formal methods in software development: Tony Hoare, Edsger Dijkstra, and Niklaus Wirth seen here at an Alpine resort.

Gates was referring to the SLAM verification engine that checks that software correctly satisfies the behavioral properties of the interfaces that it uses. The SLAM tool is now applied regularly to all Microsoft device drivers and has helped find more than three hundred bugs in the sample drivers that were supplied to developers.

Databases

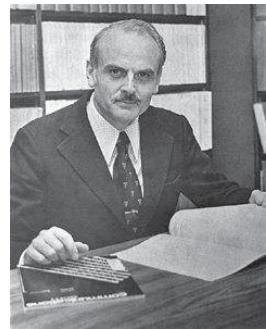
The main purpose of databases is to store and manage large volumes of data. Database software plays a vital role in our modern society. No bank transactions, online shopping, airline reservations, or even a checking out at the local supermarket would be possible without databases. Database software is now a multibillion dollar business.

The relational data model that forms the basis of modern databases is undoubtedly one of the great abstractions of the twentieth century. Historically, there were approaches for handling large volumes of data based on hierarchical, treelike structures or more general network structures. An early example of the hierarchical approach was the IBM's Information Management System. One difficulty with this approach is that not all data relationships fit well into a tree structure. A more general network structure can provide a more flexible solution, but now the user has to know the exact path leading to the data item in order to access or update it. This approach also did not scale well – with the growth of data, programmers found it difficult to navigate through a complicated web of data relationships.

The real breakthrough for database software came with the idea of the relational data model, suggested by a British mathematician Edgar “Ted” Codd (B.4.14). In 1970 he published his groundbreaking paper “A Relational Model of Data for Large Shared Data Banks.” The ideas described in this paper became the foundation of modern databases. Ironically, his own company, IBM, was initially not very supportive of his ideas. There were many skeptics and a strong resistance toward relational databases even in professional circles. In the dedication of his book *The Relational Model for Database Management* he refers to this struggle:

To fellow pilots and aircrew in the Royal Air Force during World War II and the dons at Oxford. These people were the source of my determination to fight for what I believed was right during the ten or more years in which government, industry and commerce were strongly opposed to the relational approach to database management.²⁹

The idea of a relational database is simple, yet very powerful. All the data, including the relations between data, is stored in tables that are linked together. The link is established when the same column of data is shared between two or more tables. This column is called a key. The main advantage of the relational data model is that it provides a systematic way to create the interconnections between tables. It is much easier to access data and there is no need to know the path leading to the data (B.4.15). This model is also supported by a powerful mathematical set theory and a declarative programming language called SQL – *structured query language*.



B.4.14. Edgar “Ted” Codd (1923–2003) graduated from Oxford with a degree in mathematics and chemistry and was an RAF fighter pilot during the war. After the war he joined IBM and moved to the United States. In 1981 he received the Turing Award for his contribution to the development of relational databases.



B.4.15. Jim Gray (1944–2007) on board his boat *Tenacious*. In 1988 Gray received the Turing Award for his contributions to database design and transaction processing. After gaining a PhD in computer science from Berkeley, he worked for IBM, Tandem Computers, and DEC. From 1995 Gray was a Technical Fellow at Microsoft Research. He was first to develop a website that displayed geographic data – the Terraserver – and that could deliver data to users using a web service. Gray spent the last decade of his life working with scientists with “Big Data” problems. With astronomer Alex Szalay, he pioneered the hosting of the Sloan Digital Survey astronomical data by creating the SkyServer website. Gray also coined the term *Fourth Paradigm* to reflect the increase in importance of data-intensive science. He was lost at sea, west of San Francisco Bay, in January 2007. Despite a massive collaborative effort by the emergency services and the computer science community, in searching for signs of *Tenacious*, no trace was ever found.

Design patterns

The advances of structured and O-O programming still underpin the way in which systems are written today. However, as the software industry grew, the task of teaching each new wave of programmers how to program efficiently has led to a new level of abstraction. In 1995, four software engineers – Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides – got together and identified what they called “design patterns” (B.4.16). These are standard patterns in software that everybody uses to perform a number of simple tasks.

One example is the “Observer pattern.” This ensures that when one object changes its state, all of its dependents are notified and updated automatically. They identified twenty-two patterns they called by easy-to-remember names. Apart from *Observer*, other example patterns are *Factory*, *Decorator*, *Interpreter*, and *Visitor*. Their book on design patterns has become one of the best-selling and most-cited books in computer science. It established a fixed vocabulary for talking about O-O software at a level above program code, so that programs – and programmers – became more transferable, understandable, and accurate.



B.4.16. The “Gang of Four”: Ralph Johnson, Erich Gamma, Richard Helm, and John Vlissides.

Three expensive software errors

NASA's Mariner 1 Space Probe (1962)

A bug in the flight software for the Mariner 1 (Fig. 4.14) mission caused the rocket to divert from its intended path on launch. Mission control destroyed the rocket over the Atlantic Ocean 293 seconds after launch. NASA's website says the problem was caused by a combination of two factors. Improper operation of the Atlas airborne beacon equipment resulted in a loss of the rate signal from the vehicle. The airborne beacon used for obtaining rate data was inoperative for four periods ranging from 1.5 to 61 seconds in duration. Additionally, the Mariner 1 Post Flight Review Board determined that the omission of a hyphen in the data-editing program allowed transmission of incorrect guidance signals to the spacecraft. During the periods the airborne beacon was inoperative, the missing hyphen in the data-editing program caused the computer to incorrectly accept the sweep frequency of the ground receiver as it sought the vehicle beacon signal and combined this data with the tracking data sent to the guidance computation. This caused the computer to automatically generate a series of unnecessary course corrections using the erroneous steering commands and these finally threw the spacecraft off course. The science fiction author Arthur C. Clarke wrote several years later that Mariner 1 was "wrecked by the most expensive hyphen in history."³⁰

Ariane 5 Flight 501 Launch (1996)

In his Turing Award lecture, Tony Hoare warned of the dangers of the complexities of the ADA programming language:

And so, the best of my advice to the originators and designers of ADA has been ignored. In this last resort, I appeal to you, representatives of the programming profession in the United States, and citizens concerned with the welfare and safety of your own country and of mankind: Do not allow this language in its present state to be used in applications where reliability is critical, i.e., nuclear power stations, cruise missiles, early warning systems, anti-ballistic missile defense systems. The next rocket to go astray as a result of a programming language error may not be an exploratory space rocket on a harmless trip to Venus: It may be a nuclear warhead exploding over one of our own cities. An unreliable programming language generating unreliable programs constitutes a far greater risk to our environment and to our society than unsafe cars, toxic pesticides, or accidents at nuclear power stations. Be vigilant to reduce that risk, not to increase it.³¹

Some of the ADA code for the Ariane 4 rocket was reused in the Ariane 5's control software. The error was in the code that converts a 64-bit floating-point number to a 16-bit signed integer. The faster engines caused the 64-bit numbers to be larger in the Ariane 5 than in the Ariane 4. This triggered an overflow condition that resulted in the flight computer crashing. The backup computer then also crashed, followed 0.05 seconds later by a crash of the primary computer. As a result of these software crashes, the mission was terminated thirty-seven seconds after launch (Fig. 4.15).



Fig. 4.14. Mariner 1 Space probe to Venus was the first interplanetary mission aiming to put a satellite around Venus. There are various stories about the reason why this mission had to be aborted. Most of them firmly point at a bug in the FORTRAN code of the guidance system that unexpectedly changed the trajectory of the rocket. A hyphen (overbar) missed in a mathematical expression led to the \$80 million failure. Five months later the Mariner 2 was successfully launched and completed the mission.



Fig. 4.15. Photo of the destruction of the first launch of the Ariane 5 Flight 501 rocket. Just thirty-seven seconds into the launch, the trajectory suddenly tilted by almost 90 degrees and the rocket self-destructed. The software error occurred during data conversion from a 64-bit floating-point number to a 16-bit signed integer. This led to a sequence of events that resulted in a complete loss of the guidance system.

NASA's Mars Climate Orbiter (1999)

The root cause for the loss of the Mars Climate Orbiter (Fig. 4.16) spacecraft was the failure to use metric units in the coding of the software file, “Small Forces,” used in trajectory models. Instead of reporting the thruster data in metric units of Newtonseconds (N-s), the data was reported in English units of pound-seconds (lbf-s). Subsequent processing of this thruster data by the navigation software algorithm underestimated the effect on the spacecraft trajectory by a factor of 4.45, which is the required conversion factor from force in pounds to Newtons. An erroneous trajectory was then computed using this incorrect data.

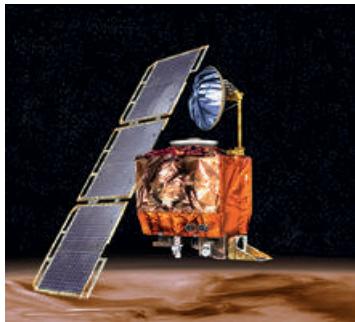


Fig. 4.16. Artist's impression of the Mars Climate Orbiter. The space probe was lost at the first attempt to enter the orbit around Mars on 3 September 1999. Putting a probe into a final planetary orbit is a long process during which the initial orbit is gradually reduced until the probe reaches its permanent orbit. Because of a software error the orbiter entered the Martian atmosphere at too high a velocity and consequently burnt up. The cost of this failure was \$125 million.

5 Algorithms

As soon as an Analytical Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will then arise – by what course of calculation can these results be arrived at by the machine in the shortest time?

Charles Babbage¹

Beginnings

What is an algorithm? The word is derived from the name of the Persian scholar Mohammad Al-Khowarizmi (see B.5.1 and Fig. 5.1). In the introduction to his classic book *Algorithmics: The Spirit of Computing*, computer scientist David Harel gives the following definition:

An algorithm is an abstract recipe, prescribing a process that might be carried out by a human, by a computer, or by other means. It thus represents a very general concept, with numerous applications. Its principal interest and use, however, is in those cases where the process is to be carried out by a computer.²

Thus an algorithm can be regarded as a “recipe” detailing the mathematical steps to follow to do a particular task. This could be a numerical algorithm for solving a differential equation or an algorithm for completing a more abstract task, such as sorting a list of items according to some specified property. The word *algorithmics* was introduced by J. F. Traub in a textbook in 1964 and popularized as a key field of study in computer science by Donald



B.5.1. The word algorithm derives from the name of a ninth-century mathematician, Mohammad Al-Khowarizmi. He was a Persian scholar who studied in the House of Wisdom, a library and research center in Baghdad. Al-Khowarizmi wrote an early text on the rules for carrying out mathematical operations with Hindu-Arabic numbers – the numbers we still use today. This book, in its Latin translation *Algoritmi de numero Indorum*, was very influential in introducing the use of Hindu-Arabic numerals and the positional representation of numbers throughout Europe. In Latin, al-Khowarizmi became known as Algoritmi, from which we get the word *algorithm*. It is also from him that we get the word *algebra* – from the Latin title of another of his books.



Fig. 5.1. A page from al-Khwarizmi's book on algebra. In this book he describes the steps for solving linear and quadratic equations and lays down the foundations of algebra as a new discipline of mathematics. The original meaning of the word *algebra* in Arabic is “to restore” – this refers to balancing out both sides of an equation. It is hard to overstate the importance of algebra in mathematics.

Knuth (B.5.2) and David Harel (B.5.3). When the steps to define an algorithm to carry out a particular task have been identified, the programmer chooses a programming language to express the algorithm in a form that the computer can understand.

The earliest known algorithm was invented between 400 and 300 B.C. by the Greek mathematician Euclid. Euclid's algorithm is a method for finding the greatest common divisor, or GCD, of two positive integers. For example, the fraction $\frac{8}{12}$ can be reduced to $\frac{2}{3}$ by dividing both numerator and denominator by their GCD, which in this case is 4. The algorithm to find the GCD of two numbers, M and N, can be expressed in four steps (see Fig. 5.2):

Step 1: Input values M, N.

Step 2: Divide M by N to find the remainder R.

Step 3: If R is zero then N is the answer, print N.

Step 4: If R is not zero, change the value of M to N and the value of N to R and go back to Step 2.

How does this algorithm work? Any number that divides both M and N must also divide the remainder R. Similarly, any number that divides both N and R must also divide M. This means that the GCD of M and N is the same as the GCD of N and R. We can see the algorithm in action in Table 5.1 for finding the GCD of 65 and 39. We begin by dividing 65 by 39 – we are interested only in the remainder, which is 26. We assign M the value of 39 and N the value of the remainder 26. In the next iteration we calculate the remainder again and assign values to M and N. We repeat the process until the remainder becomes zero, in this case the value of GCD will be held in variable N.

As Harel says in the title of his book, algorithms can be regarded as the “spirit of computing.” They are the precise procedures required to get computers to do something useful. In this chapter we will look at some examples of different types of algorithms. Historically, computers were used to solve numerical problems so we will start by looking at algorithms for numerical simulations. We will also introduce the idea of using random numbers to derive approximate answers to complex simulations. These “Monte Carlo” methods were first used in the Manhattan atomic bomb project. We will then look at problems such as finding the quickest way to sort a list of names and finding the shortest path from one city to another – as we now do routinely with our global positioning system, or GPS, navigation systems. This will lead us to a discussion of the efficiency of algorithms and an introduction to computational complexity theory.

Fig. 5.2. Flowchart of the GCD algorithm. The “%” operation calculates the remainder when M is divided by N.

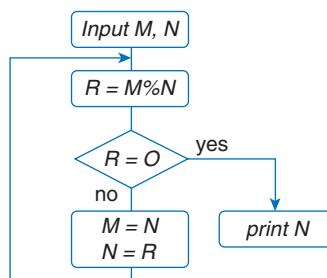


Table 5.1. Euclid's algorithm for GCD of 65 and 39

	M	N	R
Iteration 1	65	39	26
Iteration 2	39	26	13
Iteration 3	26	13	0
Result			GCD = 13



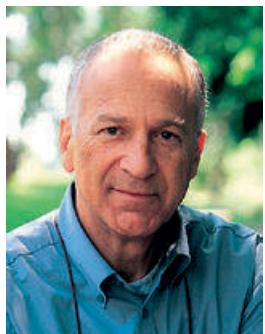
B.5.2. Donald Knuth with his series of books called *The Art of Computer Programming* has made a major contribution to the cataloging and systematic analysis of algorithms. This book is generally accepted as the “gold standard” in the field. Knuth offered a prize of one hexadecimal dollar, that is, \$2.56, for each error found in his books. These checks are considered to be trophies in academic circles. While writing his book, Knuth also developed the TeX typesetting software that is still very widely used. After winning a programming competition in the 1960s, Knuth was asked how he managed it. He replied: “When I learned how to program, you were lucky if you got five minutes with the machine a day. If you wanted to get the program going, it just had to be written right. So people just learned to program like it was carving in stone. You have to sidle up to it. That’s how I learned to program.”⁸¹

Numerical algorithms

In [Chapter 1](#) we saw that the ENIAC computer was originally built to calculate the trajectories of shells for artillery tables. In mathematics, the solution to this trajectory problem is obtained by solving the differential equation that arises from an application of Newton’s laws of motion. On a computer, such differential equations must be approximated using some numerical method.

Let’s look at a simple example. Suppose we have an object falling under the force of gravity. To find the velocity of the object at any given time, we need to solve Newton’s law for the rate of change of velocity with time. If we assume that the object is dropped from rest, we can calculate the velocity at any later time using Newton’s law in the form of a differential equation. Mathematically we can treat the velocity and time values as varying “continuously.” However, computers can only store individual “discrete” values of the velocities and time – such as the velocity after one second, the velocity after two seconds and so on. We need to approximate the differential equation by splitting up the time variable into very small increments. We can then calculate the approximate incremental change of velocity for each small increment of time. There are many different “numerical methods” that can be used to find such approximate solutions to differential equations on a computer. The simplest numerical approximation is a method devised by the Swiss mathematician Leonhard Euler ([B.5.4](#)). In practice, Euler’s method is not very precise and there are more accurate numerical methods available to solve such differential equations. [Figure 5.3](#) shows how Euler’s method compares to the exact solution for the problem of finding the velocity of an object falling under gravity through a fluid, and subject to a resistance proportional to the square of the velocity. It was the FORTRAN programming language that first gave scientists the capability of writing their programs as a relatively straightforward translation of their mathematical equations, instead of having to program the solutions to these problems using low-level machine language or assembly language.

Another important numerical technique for simulations goes by the name of the “Monte Carlo” method. In 1946, physicists at Los Alamos Laboratory were investigating the distance that neutrons can travel through various



B.5.3. David Harel gave a series of lectures on Israeli radio in 1984 explaining computer algorithms to a general audience. This led to his famous book *Algorithmics: The Spirit of Computing* and to his more recent book on computability entitled *Computers Ltd: What they really can’t do.*

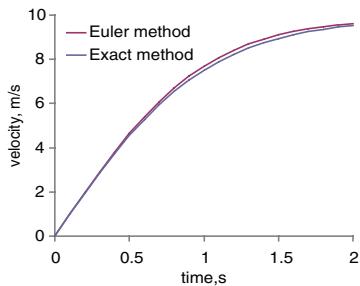


Fig. 5.3. Comparison of a numerical solution of a simple differential equation obtained using Euler's method with the exact analytical solution. If we use the Corrector-Predictor method, which is much more accurate than the simpler Euler method, we obtain results that are almost indistinguishable from the exact solution.

materials. Polish American mathematician Stanislaw Ulam ([B.5.5](#)), who, along with Edward Teller, is credited for devising a workable mechanism for the hydrogen bomb, came up with the idea of using many random experiments to find an approximate answer to the problem. He recalled his inspiration as follows:

The first thoughts and attempts I made to practice [the Monte Carlo method] were suggested by a question which occurred to me in 1946 as I was convalescing from an illness and playing solitaires. The question was what are the chances that a Canfield solitaire laid out with 52 cards will come out successfully? After spending a lot of time trying to estimate them by pure combinatorial calculations, I wondered whether a more practical method than “abstract thinking” might not be to lay it out say one hundred times and simply observe and count the number of successful plays. This was already possible to envisage with the beginning of the new era of fast computers, and I immediately thought of problems of neutron diffusion and other questions of mathematical physics, and more generally how to change processes described by certain differential equations into an equivalent form interpretable as a succession of random operations. Later [in 1946], I described the idea to John von Neumann, and we began to plan actual calculations.³

Von Neumann chose the code name Monte Carlo for the new technique in reference to the famous casino where Ulam's uncle used to like to gamble. The first unclassified paper on Monte Carlo methods, authored by Nicholas Metropolis and Ulam, was published in 1949.

We can use the Monte Carlo method to find an approximate value of π in the following way. Imagine we place a dartboard inside a square as shown in [Figure 5.4](#). If we throw darts randomly at the square, the number of darts that land within the circle is proportional to the area of the circle. By comparing the



[B.5.4.](#) Leonhard Euler (1707–83) was a Swiss mathematician and physicist. Euler was born in Basel, the son of Paul Euler, a pastor of the Reformed Church, and a friend of Johann Bernoulli, then Europe's foremost mathematician. Bernoulli recognized the young Euler's genius and convinced him to pursue mathematics rather than enter the church. At the age of thirteen, Leonhard was a student at the University of Basel and received his Master of Philosophy in 1723 for his dissertation that compared the philosophies of Descartes and Newton. After he was unable to secure a university position in Basel, in 1726 Euler was offered a post at the Russian Academy of Sciences in St. Petersburg. In 1741 he took up a position at the Berlin Academy and spent a very creative twenty-five years in Germany before returning to St. Petersburg.

Euler's mathematical abilities were supplemented by his having a photographic memory. He could recite Virgil's *Aeneid* from beginning to end and could even remember the first and last lines on each page of his edition of the book. Euler worked in almost all areas of mathematics – geometry, calculus, algebra, trigonometry, and number theory – as well as in physics and astronomy and he introduced and popularized many of the notational conventions in mathematics that we still use today. Euler was the first to write $f(x)$ to denote the function f applied to the argument x , as well as the modern notation for the trigonometric functions such as sine, cosine, and tangent; the letter e for the base of the natural logarithm; the Greek letter Σ for summations; and the letter i for complex numbers. Euler is also responsible for what physicist Richard Feynman called “the most remarkable formula in mathematics,”^{B2} Euler's identity: $e^{i\pi} + 1 = 0$.

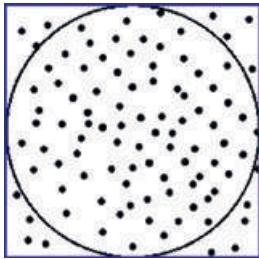


Fig. 5.4. Determination of π using Monte Carlo method. The diagram shows a circle of radius R inside a square of side $2R$. The area of circle is πR^2 and the area of the square is $4R^2$. Throwing randomly distributed darts gives an estimate of π by comparing the number of darts landing inside the circle to the number landing inside the square.

number of darts that land within the square to the number that land within the circle of radius R , we can obtain an estimate of π :

$$\begin{aligned} (\text{Number of darts inside circle})/(\text{Number of darts inside square}) &= \\ (\text{Area of circle})/(\text{Area of square}) &= \pi R^2/4R^2 = \pi/4 \end{aligned}$$

In order for this method to give an accurate value of π , we need the darts to be thrown genuinely at random, so that they cover the entire area uniformly. We also need a large number of throws. Because generating large numbers of truly random numbers is extremely difficult, von Neumann developed a clever algorithm to generate “pseudorandom” numbers on the computer. Given an initial starting number as a “seed,” these pseudorandom numbers are then generated deterministically by von Neumann’s algorithm and approximate a truly random distribution. This technique has the advantage that the exact sequence of numbers can be reproduced by starting with the same seed, and this turns out to be very helpful in debugging Monte Carlo simulation programs.

Although we have only given a very simple example here, Monte Carlo methods can be used to evaluate complex integrals in a similar manner. These methods are now widely used in many areas of science and business – and in computer algorithms for playing games.

Sorting

Although the earliest electronic computers were generally used to find numerical solutions to scientific problems, it was clear from early on that they were capable of solving other types of problems. During World War II, the Colossus computer at Bletchley Park in the United Kingdom was used for breaking codes, and soon after the war the LEO computer demonstrated the utility of computers for assisting with routine business problems, such as stock keeping, distribution, and payroll.

Let’s take a look at how computers handle such nonnumerical tasks. We will do so by examining the problem of sorting a list of names into alphabetical order. We will show how this can be done using two different algorithms,



B.5.5. Stanislav Ulam was born in 1909 in the city of Lwow in Poland, now the city of Lviv in the Ukraine. He studied mathematics at university and was a member of the Lwow School of Mathematics. The members met at the Scottish Café in Lwow and recorded their discussions in the “Scottish Book.” Ulam met John von Neumann and was invited to visit the institute at Princeton in 1935. He left Poland in 1939 just before the German invasion and many members of his family died in the Holocaust. In 1943 Ulam was an assistant professor at the University of Wisconsin in Madison and he asked von Neumann if he could join the war effort. As a result he received a letter from Hans Bethe inviting him to join the Manhattan Project, a top-secret project to build the atom bomb, based at Los Alamos, near Santa Fe, New Mexico. Because he knew nothing about New Mexico, Ulam checked out a guidebook from the university library. On the checkout slip he found the names of three colleagues who had mysteriously “disappeared” a few months before! At Los Alamos he worked on numerical solutions to the hydrodynamical equations for the plutonium implosion bomb. After the war, Ulam returned to Los Alamos to work on the development of the hydrogen bomb. In 1951, Ulam and Edward Teller came up with a mechanism for a working fusion bomb using “radiation implosion.”

called *bubble sort* and *merge sort*. Suppose we have the following list of eight names that we want to sort alphabetically:

Bob
Ted
Alice
Pat
Joe
Fred
May
Eve

Letters are usually represented in a computer using the so-called ASCII scheme, an acronym for the American Standard Code for Information Interchange. All of the twenty-six Standard English characters, plus punctuation and other symbols can be represented as a seven-bit ASCII code. Hence we can arrange for the computer to understand what we mean when we ask for two numbers to be compared and placed in alphabetical order.

Let's first examine the bubble sort algorithm. It works by repeatedly comparing adjacent names and interchanging them if they are out of alphabetical order. We start by considering the bottom two names on the list:

May	swap	Eve
Eve		May

Next we move the “bubble” up and consider the next pair on the list:

Fred	swap	Eve
Eve		Fred

We repeat the process until the bubble of paired names has reached the top. We are then guaranteed that the correct first name is at the top of the list. The detailed workings of this first iteration are shown in [Figure 5.5](#).

Now start again, with the bubble again at the bottom of the list. At the end of this second pass through all of the names on the list, the second name will be in the correct position, second from the top. For sorting all eight items correctly we need to repeat this process seven times. You can see why this algorithm is called the bubble sort, because the sorted names bubble up to the top.

The bubble sort algorithm gets the job done, but it is not a very efficient way to sort a large list. Sorting is such a common task that computer scientists have spent a lot of time looking for efficient sorting algorithms. One very clever and practical algorithm is called merge sort. It was invented by von Neumann

Fig. 5.5. An example of the bubble sort algorithm. This example works by exchanging adjacent names if they are out of order, starting from the bottom of the list, and continuing until the bubble of paired names has reached the top. This is repeated until all items are sorted correctly.

Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8
Bob	Alice						
Ted	Ted	Ted	Ted	Ted	Ted	Alice	Bob
Alice	Alice	Alice	Alice	Alice	Alice	Ted	Ted
Pat	Pat	Pat	Pat	Eve	Eve	Eve	Eve
Joe	Joe	Joe	Eve	Pat	Pat	Pat	Pat
Fred	Fred	Eve	Joe	Joe	Joe	Joe	Joe
May	Eve	Fred	Fred	Fred	Fred	Fred	Fred
Eve	May						

in 1945 and uses a fundamental technique of computer science called *divide-and-conquer*. We begin by splitting our eight-name list into two halves, so that we have two lists of four names. We then split each half again into two lists of two names. We order each of the pairs of names and then merge the sorted pairs. The merge is done by repeatedly comparing the characters at the head of each list and sending the alphabetically lower item to the output. We complete the sort by merging the two sorted lists of four names in the same way. The diagram in [Figure 5.6](#) illustrates how the merge sort algorithm works.

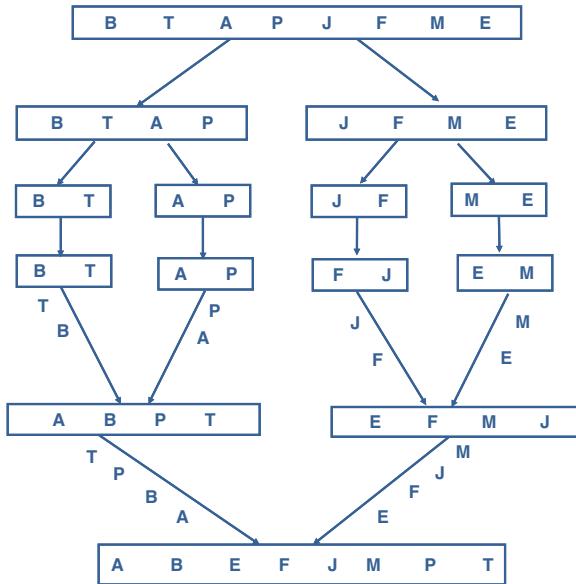
In programming the merge sort algorithm, we can write the program using “recursion” for the divide phase, creating a subroutine calling itself a number of times. In this case, we can introduce a subroutine called “Divide” and use it to split the list into two halves. If there are only two elements left in the list, it returns them in alphabetical order; if there are more than two elements, the Divide subroutine calls itself and repeats the process. This carries on until there are only one or two elements left in the divided list. The use of recursion is a very powerful programming technique much loved by computer scientists. The merge phase can also be programmed recursively using a Merge subroutine. We will compare the efficiencies of some algorithms later in this chapter, in the section on complexity theory.

Graph problems

We are familiar with the use of routing algorithms from our GPS navigation systems. Indeed the systems are becoming so reliable that we are fast approaching a time when finding our way by reading a paper map will be a lost art! All we do to find the shortest route from A to B is to enter the start and end points in the car navigation system. How do computers solve such problems? The solution uses another branch of mathematics invented by Euler: “graph theory.”

The city of Königsberg in Prussia – now Kaliningrad, Russia – was famous for a long-standing puzzle in mathematics. The city is located on both sides of the Pregel River, and there are two large islands in the river connected to the mainland by seven bridges ([Fig. 5.7a](#)). The seven bridges problem was to find a walk through the city that would cross each bridge only once. In 1735, Euler

Fig. 5.6. An example of the merge sort algorithm that uses a divide-and-conquer approach to reduce the list to sets of pairs of names. These are ordered and the different pairs merged together in the correct order.

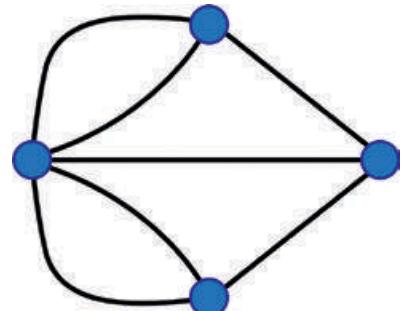
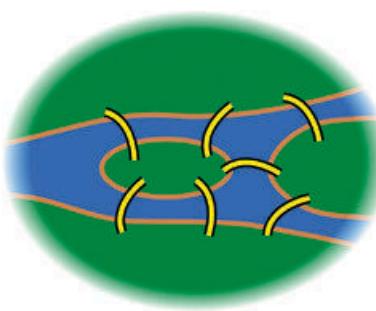
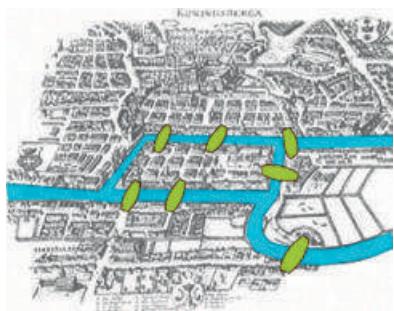


proved that there was no solution, and in so doing he laid the foundations of graph theory and the beginnings of the study of topology.

Euler solved the problem by reducing it to essentials. The choice of route on land is unimportant: only the sequence of bridges crossed is relevant (Fig. 5.7b). The map can be further simplified by replacing each landmass with a dot – called a “vertex” or a “node” – and each bridge by a line – called an “edge” – joining two vertices (Fig. 5.7c). Only the connection information in the resulting “graph” is important for this problem, not details of the layout of the figure. This illustrates one of the key ideas of topology: topology is not concerned with the rigid shape of objects or surfaces, just their connectivity.

Euler then observed that, except for the start and finish of the walk, whenever one enters a vertex (landmass) by a bridge, one must leave the same landmass or vertex by another bridge. If each bridge is crossed only once, except

Fig. 5.7. Three representations of the Seven Bridges of Königsberg: (a) Königsberg in Euler's time; (b) a more abstract representation of the seven bridges; and (c) a graph of the seven bridges.



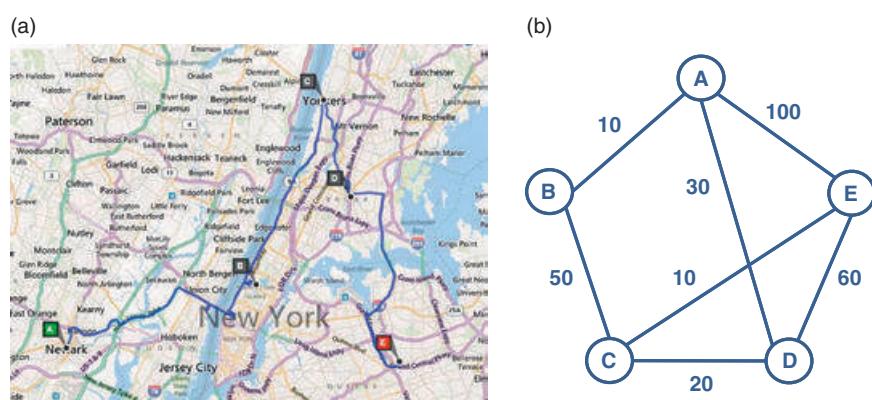
for the start and finish landmasses, the number of bridges connecting any other landmass must be an even number – half of the bridges for the walker to enter the landmass, and half for the walker to leave it. In the case of the bridges in Königsberg, we see that all the four landmasses are connected by an odd number of bridges – one by five, the other three by three. Because at most two of the landmasses can be the starting and end points, we see immediately that it is not possible to walk through the city crossing each bridge exactly once.

Let us look at another important type of graph problem. This is the problem of finding the minimal spanning tree (MST) – a path that reaches every node in a graph with the minimum cost. Consider five well-known communities in the area around New York City (Fig. 5.8a) and represent them as a graph (Fig. 5.8b). In this graph, each community is represented as a vertex, with a road joining two communities by an edge. Each edge is assigned a number representing the “cost” needed to go between the communities at the ends of each edge. This could represent the cost of a cable connection or the time of travel between the two places, for example.

Imagine that the company wants to connect its offices in the five communities using the least amount of optical fiber. The minimal spanning tree (MST) solves this problem. Finding the MST is a problem that can be solved by using a so-called greedy algorithm. Greedy algorithms take the optimal choice at each local stage of the algorithm and in general are not guaranteed to find the globally best solution but can be proved to do so for the case of the MST. In this example, we start with the shortest edge in the graph; then from the two vertices at the ends of this edge, we choose the next shortest edge. We continue to add to the resulting graph by adding the next shortest edge that has not yet been considered. We repeat this procedure until we have visited each city in the graph. The result is the MST shown in Figure 5.9.

The solution illustrates another important structure in computer science: trees. Trees are similar to graphs except that they do not contain closed loops. Trees are found everywhere in our daily lives, such as in the organization charts of companies or in the file structures on your computer (Fig. 5.10). Efficient algorithms to traverse and manipulate tree structures are an important area of algorithmics (Fig. 5.11).

Fig. 5.8. An illustration of the MST problem. The figure shows (a) five communities in the New York area: A = Newark; B = Manhattan; C = Yonkers; D = The Bronx; E = Queens; and (b) a graph of the five communities with distances allocated to each edge.



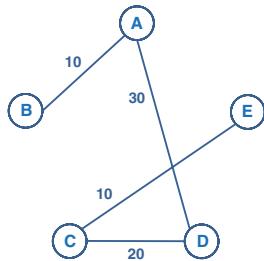


Fig. 5.9. The cheapest solution connecting all five cities with the minimum length of optical fiber is the MST for the graph in Figure 5.8.b. In this case the MST can be found using a simple greedy algorithm, as explained in the text.

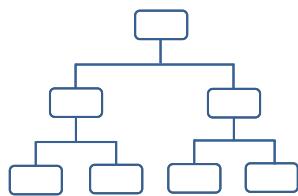


Fig. 5.10. An example of a tree structure organization. The tree data structure is one of the key concepts of computer science and they are the cornerstones of all databases. A tree consists of nodes and branches. Whenever a node is added or removed the tree needs to be adjusted in order to make it shorter and more “bushy” rather than tall and thin. This makes search operations much faster.

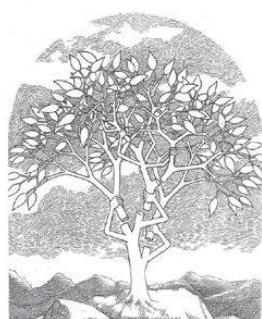


Fig. 5.11. Cartoon of a self-adjusting tree.

Let us look at another important problem. This is the problem of finding the shortest path through a graph - the sort of algorithm used by our GPS navigation systems. How does the computer embedded in our GPS system solve such problems? It does so by using a variant of the shortest path algorithm devised by Edsger Dijkstra, an early computer science pioneer in the area of programming languages and software engineering. Dijkstra was asked in an interview how he came to invent his shortest path routing algorithm and he replied:

What is the shortest way to travel from Rotterdam to Groningen? It is the algorithm for the shortest path which I designed in about 20 minutes. One morning I was shopping with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path.⁴

Let us go back to our company in the New York area. Suppose that the company's headquarters are located in Newark (city A) and that it frequently needs to deliver supplies from its headquarters to each of its offices located in the communities of Manhattan (B), Yonkers (C), the Bronx (D), and Queens (E). For simplicity's sake, let's assume that the edges of our graph are “directed” like one-way streets, meaning they can only be traveled in one direction. In addition, we need to ensure that there is no possibility of these directed edges forming a closed loop or cycle; with our one-way restrictions this is true of the graph in Figure 5.12. This type of graph occurs in many places in computer science and has the intimidating name of a Directed Acyclic Graph, or DAG.

To find the shortest path from the head office in Newark to every other office, Dijkstra's algorithm uses a greedy method. Let us see how Dijkstra's algorithm works in this case:

- The first iteration of the algorithm starts at headquarters A and finds the office that has the shortest direct connection to A. In our example of Figure 5.8b, the closest office to A is clearly B, with a distance of 10. (Note that because there is no direct connection from city A to office C, we set distance to infinity.)
- The next step in the algorithm examines the shortest paths to the other offices if we start from A as before, but also now allow the option of going through B. We see that by going through B, we can now get to C and therefore we record the distance as $10 + 50 = 60$. For the next step in the algorithm we need to add to our set of two locations, A and B, the next closest office to A. The next shortest path is now to office D, with a distance of 30.
- For the third iteration we now allow paths from A that can either go directly from A or via locations B or D. With this extra option, we see by inspecting the graph (Fig. 5.8b) that the shortest path from A to C is now through D rather than through B. Similarly, it is now shorter to get to E through D than going direct from A. Again, we complete the step by looking for the city with the next shortest path from A, which is now C with a distance of 50.

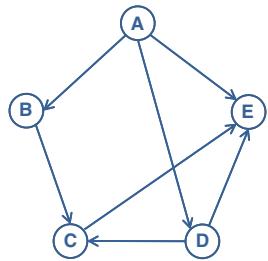


Fig. 5.12. An example of a Directed Acyclic Graph or DAG.

- For the next and final iteration, we calculate the shortest paths from A to each other location, but now allowing any of the offices B, D, and C as possible intermediate destinations. The shortest path to E is now through offices B and C rather than through D or going direct from A.

In this way, we have now found the shortest path from A to all the other locations in the graph. These iterations of Dijkstra's algorithm are summarized in [Table 5.2](#).

There are many other types of routing algorithms that can be applied to such shortest path problems. One important method is called *dynamic programming*, which is a technique that can be used when simple greedy algorithms do not give the best solution. Dynamic programming algorithms allow for long-range optimizations instead of the purely local optimizations performed in Dijkstra's algorithm.

Before we leave this section, we want to introduce an important character in the study of algorithmics and graph problems – the traveling salesman problem, or TSP, which has fascinated mathematicians and computer scientists since the 1930s. The problem can be stated as follows: given a list of cities and the distances between each of them, what is the shortest route that a traveling salesman can take to visit each city and return to his starting point?

Obviously one way of solving this problem is just to use brute force and enumerate every possible route. For our five-location network in [Figure 5.8b](#), we can calculate how many different routes the salesman could take. The problem is equivalent to finding the number of permutations of the five symbols A, B, C, D, and E. Because any shortest route starts and finishes at the same city, using any of the five cities as the starting point of the route gives the same answer. So we can just start with A and look for all the possible routes starting with A. There are then four possible choices for the second city, three for the third, and two for the fourth, before we are left with only the fifth city. Thus it looks like we need to evaluate $4 \times 3 \times 2 \times 1 = 24$ permutations (or $4!$, to use the common notation for factorials). But there is another simplification. The distance from B to C is clearly the same as the distance from C to B, and the same is true for every pair of cities. Each permutation has a reverse permutation of the same length, and it does not matter which direction we travel round the tour. We therefore need to consider only $4!/2 = 12$ different routes.

The shortest path for this problem, ABECDA, is shown in [Figure 5.13](#). Where is the difficulty with the TSP? For an N-city problem, we need to examine $(N - 1)!/2$ tours, and as the number of cities increases, this brute force

Table 5.2 Iterations of Dijkstra's algorithm. Column S is a set of cities used in the shortest path search. D[node] is the distance to a city

Iteration	S	D[B]	D[C]	D[D]	D[E]
Initial	{A}	10	∞	30	100
#1	{A,B}	10	60	30	100
#2	{A,B,D}	10	50	30	90
#3	{A,B,D,C}	10	50	30	70
#4	{A,B,D,C,E}	10	50	30	70

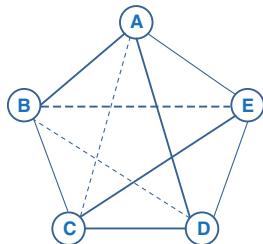
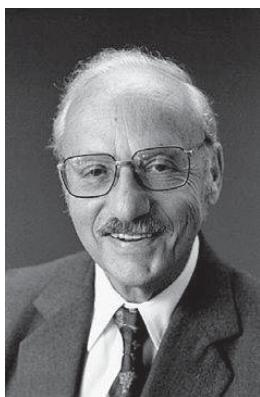


Fig. 5.13. The TSP for our five-city problem corresponds to finding the shortest round tour through all of the cities. Note that to conform to the usual formulation of the TSP problem with all-to-all paths possible between the cities, we have added in the missing “direct” paths between B and D, B and E, and A and C, taking the distances as the shortest distances to go between them, going via an intermediate node.



B.5.6. George Dantzig (1914–2005) is credited with the development of the simplex algorithm and numerous other contributions to linear programming. His algorithm is used to solve many real-life problems related to air traffic scheduling, logistics, planning processes in oil refineries, circuit design, and many more.

approach rapidly becomes impractical. We can therefore say that such a brute force algorithm for the N-city problem is *unreasonable*. To understand better what we mean by *reasonable* and *unreasonable*, we need to look at how we can measure the performance of algorithms. Before we do this, we will give a brief history of attempts at solving the TSP for large numbers of cities.

In 1954, three researchers at the RAND Corporation in Santa Monica, California – George Dantzig (B.5.6), Ray Fulkerson, and Selmer Johnson – looked at the problem of finding the shortest path for a tour through all the forty-eight contiguous U.S. states. *Newsweek* reported their success:

Finding the shortest route for a traveling salesman – starting from a given city, visiting each of a series of other cities, and then returning to the original point of departure – is more than an after-dinner teaser. For years it has baffled not only goods- and salesman-routing businessmen but mathematicians as well. If a drummer visits 50 cities, for example, he has 10^{62} (62 zeros) possible itineraries. No electronic computer in existence could sort out such a large number of routes and find the shortest.

Three RAND Corp. mathematicians, using Rand McNally distances between the District of Columbia and major cities in each of the 48 states, have finally produced a solution. By an ingenious application of linear programming – a mathematical tool recently used to solve production-scheduling problems – it took only a few weeks for the California experts to calculate “by hand” the shortest route to cover the 49 cities: 12,345 miles.⁵

The algorithm the three researchers used to solve the problem was unusual: it was just a board with pegs at the city locations and a piece of string to try out possible TSP tours. As the *Newsweek* blurb recounts, they found the shortest tour by using a powerful technique called *linear programming*. Dantzig had devised the technique as a method to schedule the training, supply, and deployment of military units when he was working at the Pentagon after World War II.

Linear programming expresses the problem as an economic model with inputs and outputs as variables subject to a set of constraints. These constraints can include inequalities, such as requiring some variables to always be greater than or equal to zero. As the name implies, the variables were combined in a set of linear equations and the goal was to choose the variables to maximize an explicit objective. To find the optimal solution to such a linear programming problem, Dantzig developed an algorithm that was named one of “The Top Ten Algorithms of the Century” in the year 2000. This is the simplex algorithm, which is still widely used in industry where the models can have hundreds of thousands of constraints and variables. A detailed discussion of this algorithm is beyond the scope of this book, but it still provides the basis for modern analyses of the TSP. Using linear programming, Dantzig, Fulkerson, and Johnson were able to prove that their solution was indeed the shortest path, writing:

In this context, the tool of choice is linear programming, an amazingly effective method for combining a large number of simple rules, satisfied by all tours, to obtain a single rule of the form “no tour through this point set can be shorter than X.” The number X gives an immediate quality measure: if we can also produce a tour of length X then we can be sure that it is optimal.⁶

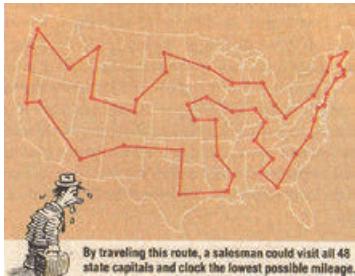


Fig. 5.14. Optimal tour around the United States visiting forty-eight state capitals. Researchers George Dantzig, Ray Fulkerson, and Selmer Johnson from the RAND Corporation did not actually use the forty-eight state capitals in their classic 1954 solution of the forty-eight-city problem.

Since the pioneering work of these RAND researchers, the challenge of the traveling salesman has continued to attract the attention of researchers. The record for finding the optimal tour has been steadily increased from 48 cities (Fig. 5.14) in 1954; to 64 by Michael Held and Richard Karp in 1971; to 532, then 1,002, and then 2,392 cities by teams led by Martin Grotschel and Manfred Padberg in 1987; to tours of 13,509 cities in the United States in 1998 and of 24,978 cities in Sweden in 2004 by Concorde, the current champion TSP program. The Concorde program was developed by David Applegate, Robert Bixby, Vasek Chvatal, and William Cook and is available over the Internet. In 2006, they used their program to find the shortest travel time for a laser to cut connections in a Bell Labs computer chip. The result was an optimal tour for an 85,900 “city” problem (Fig. 5.15). This stands as the record TSP for which the optimal tour is known. Larger problems, such as the 100,000-city Mona Lisa problem created by the artist Bob Bosch and shown in Figure 5.16, are significantly more difficult than the computer chip problem, which has many “cities” close together on straight lines. Currently the best solution for a Mona Lisa tour is still 0.0026 percent above the bound for the optimal tour!

Before we leave the traveling salesman problem we should say that although finding a provably optimal tour is still computationally challenging, there are many practical ways to find very good approximate solutions to the TSP. Most modern algorithms are variants on a method devised by Bell Labs researchers Shen Lin and Brian Kernighan in 1973. This systematizes the process of making incremental tour improvements on some initial tour. A “2-opt” move is an improvement wherein two edges are deleted and the tour reconnected with two shorter edges. Similarly, we can look for 3-opt moves and more. Danish computer scientist Keld Helsgaun improved on the original Lin-Kernighan method in 1998 by explicitly incorporating a search for 5-opt exchanges, reconnecting ten edges at a time. Combining Lin-Kernighan with ideas from simulated annealing in physics, in 1991 researchers Olivier Martin, Steve Otto, and Ed Felten at Caltech developed what is now known as the Chained Lin-Kernighan algorithm. In 2000, this method was used on a 25,000,000-city problem to find a tour that was only about 0.3 percent greater than the theoretical shortest path. This is still the dominant algorithm for use with very large data sets. The TSP is an important optimization problem for many types of problems – from various pickups and deliveries, to finding markers on genomes, to moving telescopes and manufacturing electronic circuit boards.

Complexity theory

As Charles Babbage foresaw in the quotation that introduces this chapter, now that we have computers, the question of how to find the fastest algorithm to solve a particular problem moves to center stage. In our discussion on sorting

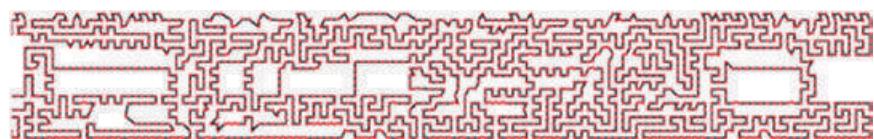


Fig. 5.15. Section of the optimal tour for the 85,900-“city” problem.



Fig. 5.16. In 2009, Robert Bosch from Oberlin College generated a set of 100,000 points and then ran the TSP algorithm on this set in order to calculate the minimal path. As the algorithm proceeds it connects the dots with lines and the outcome resembles Leonardo da Vinci's enigmatic painting of the Mona Lisa.

we looked at two different algorithms – bubble sort and merge sort – and we claimed that merge sort was much more efficient than bubble sort. How can we justify such a statement? This type of question is the business of complexity theory, which examines the computational resources required by an algorithm or class of algorithms. Typically these resources are measured as *time* (the number of computational steps required to solve the problem) or *space* (how much memory does it take to solve the problem). Let us look at the time complexity of our sorting algorithms.

How many operations do we require to sort N objects according to both algorithms? In the bubble sort algorithm, we have to go through the entire list of N objects and perform $(N - 1)$ comparisons. We then have to repeat this process $(N - 1)$ times. To sort a list of length N , we see that for the bubble sort algorithm, the number of comparisons we are required to carry out is:

$$(N - 1) \times (N - 1) = N^2 - 2N + 1$$

Of course there are other statements in the program besides these comparisons, but we are only interested in the behavior of the algorithm for large N . In this case, it is safe for us just to look at the comparisons because the other parts of the program – involving testing and manipulating indices, for example – just take a fixed amount of time. In addition, because for large N , the N^2 term is much larger than the $(-2N + 1)$ term, we can say that the amount of computational work in the bubble sort algorithm applied to N objects grows approximately like N^2 . Complexity theorists write this behavior as $O(N^2)$, where the “big-O notation” specifies how the running time of the bubble sort algorithm grows with N .

What about the time complexity behavior of the merge sort algorithm? In this case we used a divide-and-conquer approach and we do not have to cycle through the entire list multiple times. For merge sort, we divide the list up by repeatedly dividing N by 2 and then make comparisons on just the multiple lists containing 2 items. How many times can we divide a list of length N ? In our example, we started with 8 items and went from 8 to 4 to 2 so there were three layers and two calls to subroutine Divide. Note that $8 = 2^3$ and we can write the number of layers in terms of logarithms to the base 2. With our more familiar base 10 logarithms, we can write the power of 10 in 1000 as the logarithm $\log_{10} 1000 = 3$. Similarly, we can write the numbers of divisions by 2 for 8 items as the logarithm to base 2, namely $\log_2 8 = 3$ (very handy for binary machines like computers). In general, for N elements we can write the number of divisions as $\log_2 N$.

Because the number of divisions grows like $\log_2 N$ and the number of comparisons we need to make grows like N , the complexity of the merge sort algorithm is $O(N \log_2 N)$. This is the beauty of the divide-and-conquer approach. **Table 5.3** shows how the growth rates of N^2 and $N \log_2 N$ compare. We see that $N \log_2 N$ grows much more slowly with N than does N^2 – thus showing the importance of a good sorting algorithm. Any algorithm whose time complexity grows slower than some polynomial – in this case $N \log_2 N$ grows slower than N^2 – is said to be *reasonable*. Any problem for which we can find a low-order polynomial time algorithm is said to be *tractable*, meaning that it can be evaluated by a computer in an acceptable amount of time.

Table 5.3 Growth of operations for sorting algorithms				
N	10	50	100	300
$N \log_2 N$	33	282	665	2469
N^2	100	2500	10000	90000

Now let us go back to the traveling salesman problem. We have seen that the brute force method to find the exact solution for the shortest path through N cities grows like $N!$. A factorial grows with N much faster than any polynomial. As we have seen, we can do better than this brute force solution. Using dynamic programming, in 1962 Held and Karp found an algorithm that solves an N -city TSP in a time proportional to $N^2 2^N$. 2^N corresponds to an exponential time complexity. Exponential growth occurs when the rate of growth of a function is proportional to its current value. As can be seen from Figure 5.17, exponential growth rapidly outstrips linear and quadratic growth, and in fact outstrips any polynomial growth. This means that even though this algorithm to find an exact solution for the traveling salesman problem is much better than our brute force method, it is still unreasonable in that any computational solution will take a time that grows exponentially with N . Any problem for which we can find only exponential time algorithms is said to be *intractable*.

Does P = NP?

Before we leave the subject of algorithmics and complexity, we must introduce one of the most difficult unresolved problems in computer science. It turns out that the traveling salesman problem is representative of a large class of problems that have unreasonable, brute force solutions but for which it cannot be proved whether much faster, reasonable, algorithms exist. These problems are as diverse as devising a timetable to allocate teachers and courses to classrooms with all sorts of constraints; packing items of varying sizes and shapes into fixed-size bins; and determining possible arrangements of patterned tiles. Finding some acceptable solution to even small versions of these problems in real life usually involves much trial and error. After we have made a choice that seemed to be the best possible choice at the time it turns out not to be and we have to backtrack and try some other choice. All of these problems have exponential time solutions, and no one has been able to find an algorithm that solves any of these problems in polynomial time.

The problems in this class are called *NP-complete*. Computer scientists denote the class of all problems that are tractable and have algorithms that take only polynomial time by the symbol P. Besides only having known exponential time solutions, the NP-complete problems have two other important properties: they are nondeterministic, which is what N stands for, and they are complete. To understand what these terms mean let us return to the traveling salesman and pose the problem slightly differently by asking whether or not we can find a tour with a length shorter than a given number of miles. As we have seen, it is very difficult to find the shortest tour, but if we are given a specific tour, it is very easy to verify whether this tour is shorter than the specified length. Where does the nondeterminism come in? Suppose we are trying to find the shortest tour and start out at some city. There are some obvious possibilities for the first step so we toss a coin to decide which city we should visit first. If there are more than two cities to choose from we will have to toss the coin more than once. We now suppose that the coin is not a normal one that just gives a random result, but a “magical” one that always leads to the best choice. The technical term for this magic is *nondeterminism*, and it means that

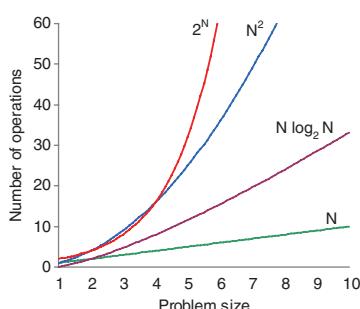
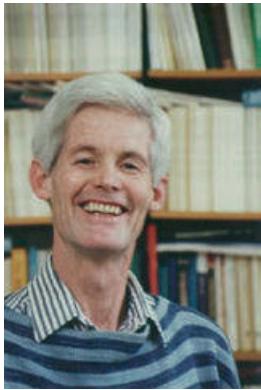


Fig. 5.17. This graph shows the growth with problem size N of four different functions: N ; $N \log_2 N$; N^2 ; and 2^N . The growth of an exponential function like 2^N is much faster than any polynomial like N^2 .



B.5.7. Steven Cook received the Turing Award in 1982 for his contribution to algorithmic complexity research.

we do not have to try all the choices to find the right solution. As we can see from the fact that it is easy to check whether or not we have a correct solution, this nondeterministic method can find the solution in polynomial time. This is why these problems are called NP – since there is a nondeterministic polynomial solution.

The second property of NP-complete problems is perhaps the most remarkable. No one has been able to prove that there does not exist a polynomial time algorithm for any of these problems. What the designation *complete* signifies is that if a polynomial time solution were found for one of these problems, then there would be a polynomial time algorithm for all of them! How does this come about? Let us look at another path-finding problem, one that does not involve distances. If we are given a graph consisting of points and edges, can we find a path that passes through all the points exactly once? Such a path is called a *Hamiltonian path*, after the great Irish mathematician William Hamilton. Figure 5.18a shows a Hamiltonian path through five nodes. This problem also turns out to be intractable and NP-complete. Curiously, if we want a path that goes through all the edges exactly once – called an *Eulerian path*, as in Euler's solution to the Bridges of Königsberg problem – the situation is very different. Euler found a polynomial time algorithm for this problem in 1736!

As we have said, the complete in NP-complete signifies that all the problems stand or fall together. Either all NP-complete problems are tractable or none of them are. The concept that is used to establish this is to show that there is a polynomial time algorithm that reduces one NP-complete problem to another. We can see how this works by reducing the Hamiltonian path problem to the traveling salesman problem. In Figure 5.18a we have a graph with five nodes and we have highlighted the Hamiltonian path for this graph. We can construct a traveling salesman network from this graph by using the same nodes, but also drawing additional edges connecting every two nodes as in Figure 5.18b. We assign cost 1 to an edge if it was originally present and cost 2 for each new edge we have added. The new graph has a traveling salesman shortest path of length 6 units – in general $N + 1$ where N is the number of nodes in the graph – if the original graph had a Hamiltonian path. Thus the answer to whether or not there is a tour no longer than $N + 1$ is the same as asking whether or not the graph contains a Hamiltonian path. Since the

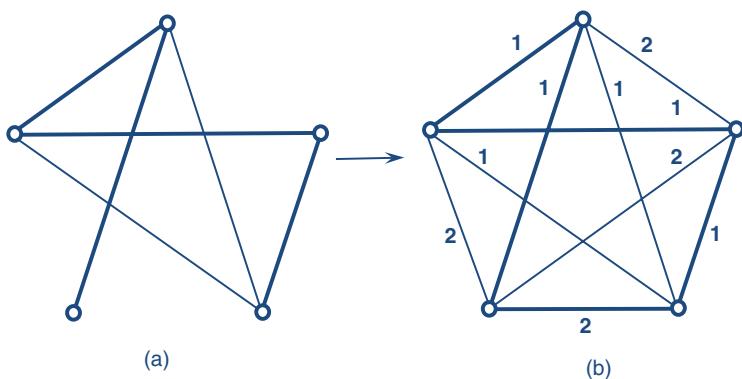


Fig. 5.18. (a) The Hamiltonian path (in bold) connecting five nodes goes through each node exactly once. (b) The Hamiltonian path problem can be converted into a TSP by adding extra edges as described in the text. The traveling salesman tour is shown in bold. (Figure courtesy of David Harel.)

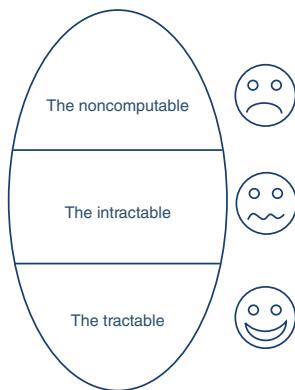


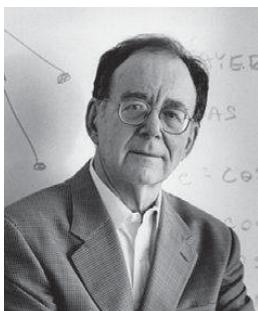
Fig. 5.19. This figure from David Harel's book shows the main problem categories: noncomputable problems have no algorithmic solution. Algorithms for intractable problems do exist but only with exponential or higher order of complexity: tractable problems can be solved with polynomial time algorithms.



B.5.8. Leonid Levin discovered the class of NP-complete problems independently from Stephen Cook.

Algorithmics and computability

Numerical simulations of complex physical systems are still a major application area for today's computers. For problems that are very complex, such as weather forecasting or global climate modeling, scientists need to use the fastest, most expensive machines – supercomputers with multiple processors. However, we have also seen how computers can be used to address a variety of different types of problems, from sorting to graph problems. It is here that we have seen the need to use clever algorithms that enable us to solve these problems as quickly as possible. But we have also seen that there are some problems



B.5.9. In 1972 Richard M. Karp wrote a groundbreaking paper, "Reducibility among Combinatorial Problems," in which he identified twenty-one combinatorial problems belonging to the class of NP-complete problems that can be reduced to a common problem – the so-called satisfiability problem. In 1985 he received the Turing Award for his contribution to algorithmic research.

for which no reasonable algorithms exist: the traveling salesman problem is just one of a number of problems for which we know of no polynomial time algorithm. In the next chapter we shall see that there are not only tractable and intractable problems, but also those that are *noncomputable* by any algorithm or computer!

Key concepts

- Algorithms as recipes
 - Euclid's algorithm
- Numerical methods
 - Discrete approximation to continuous variables
 - Monte Carlo method and pseudorandom numbers
- Sorting algorithms
 - Bubble sort
 - Merge sort
- Graph problems
 - Minimal spanning tree
 - Dijkstra's shortest path algorithm
 - Traveling salesman problem
- Complexity theory
 - Big-O notation
 - Polynomial time, tractable problems
 - Exponential time, intractable problems
 - NP-complete problems



6 Mr. Turing's amazing machines

Electronic computers are intended to carry out any definite rule of thumb process which could have been done by a human operator working in a disciplined but unintelligent manner.

Alan Turing¹

WARNING: This chapter is more mathematical in character than the rest of the book. It may therefore be hard going for some readers, and they are strongly advised either to skip or skim through this chapter and proceed to the next chapter. This chapter describes the theoretical basis for much of formal computer science.

Hilbert's challenge

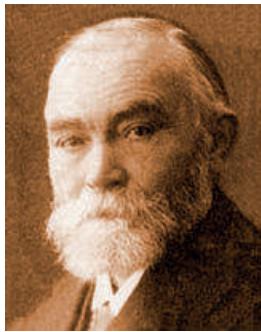
Are there limits to what we can, in principle, compute? If we build a big enough computer, surely it can compute anything we want it to? Or are there some questions that computers can never answer, no matter how big and powerful a computer we build. These fundamental questions for computer science were being addressed long before computers were built!

In the early part of the twentieth century, mathematicians were struggling to come to terms with many new concepts including the theory of infinite numbers and the puzzling paradoxes of set theory. The great German mathematician David Hilbert (B.6.1) had put forward this challenge to the mathematics community: put mathematics on a consistent logical foundation. It is now difficult to imagine, but in the early twentieth century, mathematics was in as great a turmoil as physics was at that time. In physics, the new theories of relativity and quantum mechanics were overturning all our classical assumptions about nature. What was happening in mathematics that could be comparable to these revolutions in physics?

In the late nineteenth century, mathematics was becoming liberated from its traditional role in just having application to counting and measurement. In high school algebra, letters started to be used as symbols for numerical quantities. By the twentieth century, a more abstract view had emerged in which an “obvious” numerical rule of algebra such as $x + y = y + x$, was taken to be just a rule about how symbols could be moved around, and not necessarily to be interpreted in terms of numbers. Hilbert was one of the leaders of the formalistic approach to mathematics, which transformed mathematics into a



B.6.1. David Hilbert (1862–1943) believed that in mathematics all problems could be solved, provided we work hard enough. The writing on his tombstone reads “Wir müssen wissen, wir werden wissen” – or in English “We must know, we will know.”



B.6.2. Gottlob Frege (1848–1925) was working on the axiomatization of mathematics trying to derive a logical system that is complete and has no contradictions. His pioneering work had a significant impact on later discoveries in mathematics.



B.6.3. Bertrand Russell (1872–1970) mathematician and philosopher. With Alfred Whitehead in 1910 he published *Principia Mathematica* with more than a thousand pages in which they tried to put mathematics on solid foundations. They soon learned that mathematics is not “perfect” because there are many paradoxes that mathematics cannot answer.

more abstract formulation that allowed the exploration of new kinds of algebra with different rules and symbols. This led to the development of new fields of research such as “group theory” and “Hilbert spaces,” both of which ultimately found application in physics. However, attempts to show that the whole edifice of this more formal approach to mathematics was consistent and free from contradictions had run into trouble. The key challenge was to show that manipulating the symbols according to the agreed rules always made sense and did not lead to contradictions such as being able to prove that $2 + 2 = 5$.

The German logician Gottlob Frege (B.6.2) and the Welsh mathematician Bertrand Russell (B.6.3) had independently approached the problem of proving the consistency of mathematics using the ideas of set theory. A set is just a collection of objects characterized by a particular property. For example, we can define a set that contains all the men in a town. We can also define a subset of this set that comprises all the men in the town with red hair, and so on. However, in 1901, Russell noticed that logical contradictions arose when he tried to use “sets of all sets” in his arguments. He explained his paradox with the example of a town that had only one male barber and where all the men were clean shaven (Fig. 6.1). The paradox can then be stated as: *The barber in this town shaves only men who do not shave themselves*. In this case, we have the set of all men in the town, and within this set there are two subsets – men who shave themselves and men who are shaved by the barber. The paradox is into which subset do we put the barber? Since these sets were taken to be abstract entities there was no way of resolving the contradiction by asking what the symbols really meant. The whole idea of Frege and Russell’s program was “to derive arithmetic from the most primitive logical ideas in an automatic, watertight, depersonalized way.”² Russell wrote a letter about this paradox to Frege in 1902 and Frege’s reply gives some idea of the consternation that Russell’s letter caused:

Your discovery of the contradiction has surprised me beyond words and, I should almost like to say, left me thunderstruck, because it has rocked the ground on which I meant to build arithmetic. Your discovery is at any rate a very remarkable one and it may perhaps lead to a great advance in logic, undesirable as it may seem at first sight.³

By 1928, the focus had moved on from Frege and Russell’s ambitious attempts to determine what mathematics really was. Instead, Hilbert was asking deep questions about the logical foundations of mathematics. In 1899 he had succeeded in finding a set of axioms – a small number of self-evident truths – from which he could prove all the theorems of Euclidean geometry without any need to relate his proofs to the geometry of the actual physical world. A year later, at a conference in Paris, Hilbert proposed a list of twenty-three important unsolved mathematical problems. The number two question on his list was “the compatibility of the arithmetical axioms.” Of the many questions that could be asked about these axioms, he said that the most important question was:

To prove that they are not contradictory, that is, that a definite number of logical steps based upon them can never lead to contradictory results.⁴



Fig. 6.1. Who is going to shave the barber? Illustration of barber's paradox.

Many years later, at the Bologna International Conference of Mathematicians in 1928, Hilbert made this question more precise. In Andrew Hodges's words, Hilbert's three questions were the following:

First, was mathematics complete, in the technical sense that every statement (such as “every integer is the sum of four squares”) could either be proved or disproved? Second, was mathematics consistent, in the sense that the statement “ $2+2 = 5$ ” could not be arrived at by a sequence of valid steps of proof? And thirdly, was mathematics decidable? By this he meant, did there exist a definite method which could, in principle, be applied to any assertion, and which was guaranteed to produce a correct decision as to whether that assertion was true.⁵

This last question came to be known as the decision problem – more often known by its more intimidating German name as the *Entscheidungsproblem*. Hilbert clearly believed that the answer to all of these questions would be “yes” – but within a few years, Kurt Gödel (B.6.4) had dealt Hilbert’s newly announced program a fatal blow.

Gödel was known for being extremely meticulous – in his secondary school he was famous for never making a single grammatical error. He went to the University of Vienna in 1924 and wrote his famous paper on the incompleteness of mathematics in 1931. In that paper Gödel proved a startling result. He showed that mathematics must be incomplete – in that there are statements that can neither be proved nor disproved starting from a given set of axioms. In order to prove this result, Gödel first showed how the rules of procedure of any formal mathematical system, and the use of its axioms, could be encoded as purely arithmetical operations. Having done this, Gödel was able to reduce things like the property of “being a proof” or of “being provable” to arithmetic statements. He could then construct arithmetical statements that referred to themselves, rather like Russell’s use of “sets of all sets” arguments. In particular, Gödel was able to construct a mathematical statement that effectively said “This statement is unprovable.” The statement cannot be proved *true*, for this would immediately be a contradiction. But similarly, it cannot be proved *false*, because this also leads to a contradiction. This is reminiscent of the famous liar paradox: when a man says “I am lying,” is he telling the truth or not?

Gödel was also able to show that arithmetic could not be proved to be consistent with just the use of its own axioms. In one remarkable paper, Gödel had answered the first two questions of Hilbert’s program in the negative! This left



B.6.4. Kurt Gödel (1906–78) with Albert Einstein in Princeton. John von Neumann had a very high respect for Gödel and at Princeton he helped him to get a permanent position. Allegedly he said “How can any of us be called professor when Gödel is not?”^{b1} Despite their age difference, Einstein and Gödel were close friends. They used to walk together to the Institute for Advanced Studies every morning and toward the end of his life, Einstein remarked that “his own work no longer meant much, that he came to the Institute merely to have the privilege of walking home with Gödel.”^{b2}

just the last question – Hilbert's *Entscheidungsproblem* (the decision problem). Enter Alan Turing and his amazing machines.

Turing machines

At the age of nineteen, Turing (B.6.5) went to King's College, Cambridge in 1931 to study mathematics. He passed his final mathematics examinations at Cambridge in 1934 with a distinction and was awarded the title of "Wrangler," a title still used at Cambridge to denote the top mathematics students each year. In the spring of 1935, Turing attended a lecture course given by Max Newman on "The Foundations of Mathematics." Unusual for mathematicians at Cambridge at that time, Newman was an expert in the emerging field of topology and had also followed the progress of mathematical logic and set theory since the efforts of Frege and Russell. In particular, Newman had attended the 1928 international congress at which Hilbert had announced his challenge to



B.6.5. Computer-generated image of Alan Turing. Alan Mathison Turing (1912–54) was one of the founders of computer science. His name is mainly associated with Turing machines, universality, the Church-Turing thesis, and artificial intelligence and the Turing Test. After attending Sherbourne "public school" – in England this means a private school – Turing went to King's College, Cambridge in 1931 to study mathematics. He was twenty-four when he wrote his groundbreaking paper, "On Computable Numbers, with an Application to the Entscheidungsproblem." Turing was a good long-distance runner – his best time for the marathon was only eleven minutes slower than the winning time at the 1948 Olympics – and, for recreation, he liked to go running in the countryside around Cambridge. He said later that he conceived the idea of how to answer Hilbert's third question while lying in the meadow at Grantchester, a village near Cambridge, at a break in one of his runs. It is no exaggeration to say that this paper is one of the cornerstones of computer science.

During the war he worked on code breaking at Bletchley Park, for which he was honored as an Officer of the British Empire. After the war, Turing returned to his ideas of building a physical realization of his abstract machine. At the U.K. National Physical Laboratory (NPL) in 1945 he designed the Automatic Computing Engine (ACE) which could have been the first stored-program computer. Because of bureaucratic delays for the ACE project, Turing became frustrated and left NPL. In 1949 he started to work at the computing laboratory in Manchester where he developed programs for the Manchester Mark I computer.

The following year Turing published the paper "Computing Machinery and Intelligence" in which he speculated about whether computers can think. In this paper he described what has become known as the Turing test. This is a purely operational definition of intelligence. A human interrogator poses questions to a closed room containing either a computer or a person. If the interrogator is unable to tell which is the computer and which is the person from the responses the computer is deemed to have Turing's test for intelligence.

In 1952 homosexuality was still illegal in Britain and Turing was charged with committing a homosexual act by Manchester police. As an alternative to prison Turing opted for hormone therapy which had some unpleasant side effects. Turing died in 1954 after eating an apple containing cyanide; an inquest ruled his death to be suicide. Details of his school days, his foundational research on computability, his work on the German Enigma machine at Bletchley Park, and the tensions caused by his homosexuality are contained in a wonderful biography by Andrew Hodges, *Alan Turing: The Enigma of Intelligence*. In September 2009, after an Internet campaign, the then British prime minister Gordon Brown issued an official public apology "for the appalling way he [Turing] was treated." Finally on the 24 December 2013, Turing was given a posthumous pardon by the Queen.

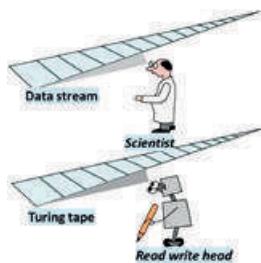


Fig. 6.2. Turing designed his machine to mimic the behavior of a human computer.

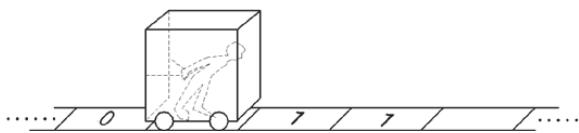
clarify the foundations of mathematics. Newman's 1935 lectures finished with an account of Gödel's theorem and made clear that the third of Hilbert's questions – the *Entscheidungsproblem* – remained unanswered. In his biography of Turing, Hodges highlights the question asked by Newman that started Turing on his journey to Turing machines in the following way:

Was there a definite method, or as Newman put it, a *mechanical process* which could be applied to a mathematical statement, and which would come up with the answer as to whether it was provable?⁶

By using the phrase “mechanical process,” Newman probably meant nothing more than an algorithm – a set of detailed instructions that leads to the solution of a problem. However, the phrase must have struck a chord with Turing. He decided to work on Hilbert's problem that summer but characteristically did not ask Newman for advice or even tell him about his intentions, nor did he read up on all the available research literature. This isolation from any of the accepted modes of thinking about the problem was undoubtedly one of the reasons that Turing followed such a strikingly unconventional approach to Hilbert's problem.

Turing equated the concept of “computability” with the ability of a very simple machine to perform a computation. His idea was to imagine a machine that worked like a human “computer” who just had to follow a set of rules (Fig. 6.2). Instead of a standard sheet of paper, Turing idealized his computer – who we shall take as female – as using a long strip of paper or “tape” on which to do her calculations. The tape is broken up into square “boxes,” in each of which she can read or write a symbol. Using a tape with only one row of symbols to do the calculations – rather than a piece of paper that could accommodate multiple rows – would undoubtedly be a very tedious restriction for a real human computer but it is perfectly possible to arrange to perform the calculation in this way. Turing imagined that his human calculator would have a number of different “states of mind” that tell her how she should use the information that she reads in each box of the tape. Thus our female computer starts off in one specific state of mind and examines the content of one of the boxes on the tape. After reading the symbol in the box, she can overwrite the symbol in the box, change to a new state of mind, and move to consider the symbol in the next square – to the left or the right. It is her state of mind that tells her what to do with the symbol she has read – whether it should be used as part of the process of addition or of multiplication, for example. Turing envisaged that a human computer would need only a finite number of states of mind to complete any given calculation. Having broken down how a human would actually go about performing the calculation into these simple steps, Turing then proposed a very simple machine that could mimic all the actions of the human computer and so work through the algorithmic steps to complete the same calculation (Fig. 6.3).

Fig. 6.3. This figure illustrates the concept of a Turing machine as “a human in a box.” The box has no bottom so the human can read the symbol under the box.



To summarize, Turing machines were to be provided with paper in the form of a tape. The tape is marked off into boxes and each box can contain at most one symbol (Fig. 6.4). At each step of the algorithm, the head of this “super-typewriter” machine can move one space, to the adjacent box on the left or on the right. The paper tape is assumed to be unlimited in length so that although the machine has a finite number of symbols and states, it is allowed an unlimited space for its calculations. This is not to say that the amount of paper attached to such a machine actually is infinite. At any given stage in any calculation the length of tape will be finite but we have the option of adding more tape when we need to. Turing’s machine is therefore able to read and write, move left or right along the tape, as specified by its set of states. The action of the machine is simple: it starts off in a certain state and looks at the contents of the first box. Depending on the state and the box contents, it will either erase the contents of the box and write something new, or leave the box as it is. Whatever it does, it next moves one box to the left or the right and changes to a new internal state. A simple example of a “Parity Counter” Turing Machine – which determines if the number of 1s or 0s in a binary string is even or odd – is described in detail at the end of this chapter.

Computable numbers and computability

With this simple machine, Turing was able to define what was meant by “computability.” To illustrate this we shall look at the question of “computable numbers.” We begin by defining what we mean by the term *real numbers*.

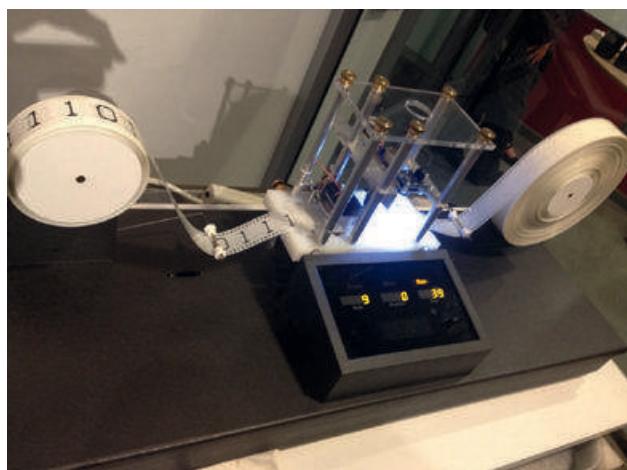
The *natural numbers* are the whole numbers (no fractions or decimal points) starting from zero:

$$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 \dots$$

Natural numbers can be added or multiplied together to produce new natural numbers. If we want to allow for subtraction, however, we need to include negative numbers as well. We define the *integers* as:

$$\dots, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7 \dots$$

Fig. 6.4. The photograph shows a working model of a Turing machine.



If we also want to include division, the integers are still too limited. We need to include fractions or *rational numbers*:

$$0, 1, -1, \frac{1}{2}, -\frac{1}{2}, 2, -2, \frac{3}{2}, -\frac{3}{2}, \frac{1}{3}, -\frac{1}{3}, \dots$$

The rational numbers are nice and neat but they leave out important quantities such as π or $\sqrt{2}$. These are called *irrational numbers* because they cannot be expressed as integers or fractions of integers. Both π and $\sqrt{2}$ can only be expressed as infinite series – as a sum of an infinite number of terms. In practice, for a useful approximate value of π or $\sqrt{2}$, we need only to sum up a few terms of the series. For example, for π , we could use the so-called Gregory-Leibniz expansion:

$$\pi = 4(1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 + \dots)$$

and for $\sqrt{2}$, we could use the Taylor expansion

$$\sqrt{2} = 1 + 1/2 - 1/(2 \times 4) + (1 \times 3)/(2 \times 4 \times 6) - (1 \times 3 \times 5)/(2 \times 4 \times 6 \times 8) + \dots$$

There are many other methods for calculating π and for taking square roots. All these methods lead to the well-known decimal approximations for π and $\sqrt{2}$

$$\pi = 3.14159265 \dots$$

and

$$\sqrt{2} = 1.41421356 \dots$$

In the struggle to understand what could and could not be proved, the question arose of what numbers could be calculated. This led to the concept of an “effective procedure” – a set of rules telling you, step-by-step, what to do to complete a calculation. In other words, if there is an effective procedure for some computational problem it means that there is an algorithm that can be executed to solve the problem. These methods for calculating π and $\sqrt{2}$ are examples of effective procedures. They may not be the most efficient way to calculate π or $\sqrt{2}$ but these algorithms will work and will produce an answer.

The number system that includes irrational numbers like these is the system of *real numbers*. In everyday life, we use approximations to real numbers and do our calculations accurate to a specific number of decimal places.

How many real numbers are there? Georg Cantor (B.6.6), who developed the theory of infinite numbers in the late 1800s, showed that the number of integers is the same as the number of natural numbers. He did this by setting up a one-to-one correspondence as follows:

Integers	0	-1	1	-2	2	-3	3	-4	...
Natural numbers	0	1	2	3	4	5	6	7	...

Although it may seem that there are more integers than natural numbers, Cantor showed that the integers could in principle be counted off against the natural numbers in this way. Although both were infinite, the existence of such



B.6.6. The name of Georg Cantor (1845–1918) is associated with set theory and with tackling the problem of infinity in a mathematically rigorous way. Cantor came to the conclusion that the infinite set of real numbers is larger than the infinite set of natural numbers. Furthermore, he was able to show that there is an infinite number of infinities. Cantor's ideas met with considerable resistance from fellow mathematicians. The great German mathematician, David Hilbert, was an exception and was early to recognize the significance of Cantor's work. Hilbert later said: "No one shall expel us from the Paradise that Cantor has created."⁸³

a one-to-one correspondence in Cantor's theory of infinities establishes that in a technical sense the number of objects in the top row is the *same* as the number of objects in the bottom row. Thus the number of integers is the same as the number of natural numbers, although both are infinite. Sets that can be put into one-to-one correspondence with the natural numbers are said to be "countable." Similarly, we can arrange for all the rational numbers to be in a one-to-one correspondence with the natural numbers so they, too, are countably infinite. But what about the real numbers? Here the situation is very different and Cantor proved this using his "diagonal slash" method that was later used by both Gödel and Turing. Using this technique Cantor was able to show that the number of real numbers must be greater than the number of natural numbers and is therefore not countable.

Let us see how the technique works. We begin by assuming the opposite – namely that we can pair off the real numbers with the natural numbers in some way. We make a list of all the real numbers we can think of and associate each decimal number with a natural number as follows:

Natural	Real
0	0.124 ...
1	0.0 <u>1</u> 5 ...
2	0.53 <u>6</u> 92 ...
3	0.800 <u>2</u> 444 ...
4	0.3341 <u>0</u> 5011 ...
5	0.34256 <u>7</u> 8 ...

The exact assignment of real numbers to the natural numbers is arbitrary: all we need to do is to assign one real number per natural number so that all the real numbers are accounted for. But this cannot be so! To see why, Cantor showed how to find another real number that cannot be already on our list. In the preceding list, we underline the first digit of the first number, the second digit of the second, the third of the third, and so on. This gives us the sequence:

$$1, 1, 6, 3, 0, 7, \dots$$

The diagonal slash procedure is to construct a new real number from this sequence that differs from the digits of this number in each corresponding place. We make this new number by ensuring that the n th digit of this new number differs from the n th digit in this sequence. For example we can define a new real number by adding one to each of the underlined digits (with the rule that $9 + 1 = 0$) to get:

$$0.227418 \dots$$

What have we achieved? By construction, this number differs from the first number in the first decimal place, from the second number in the second place, from the third number in the third place, and so on. By construction this number is different from any of the real numbers on our original list. Hence we have found a real number that cannot be on our list. This contradiction establishes the fact that there cannot be a one-to-one correspondence

between the real numbers and the natural numbers and that the real numbers are not countably infinite.

How does this argument relate to Turing machines and computable numbers? We can obviously construct a machine to calculate the decimal expansion of π by using one of the many algorithms for determining π . This just requires a set of rules for adding, multiplying, and so on. However, because π is an infinite decimal, the work of the machine would never end and the machine would need an unlimited amount of working space on its tape. A legitimate Turing machine must halt, so we need to set up the machine to calculate each successive decimal place of π as a separate calculation. Each number in the decimal expansion would then use only a finite amount of tape and take some finite time for the machine to compute. So a Turing machine for producing the decimal expansion of π to any number of decimal digits does exist in this sense, although it would be a little complicated to set up. And we can obviously do the same for a real number like the square root of two. The real numbers that can be generated in this way Turing called “computable numbers.”

In his paper Turing showed that the number of his machines was countable. To see this, we specify any given Turing machine by the “quintuple” description of the machine as we see in our detailed discussion of the Parity Counter Turing Machine at the end of this chapter. The quintuple description is just an explicit labeling of the actions of the Turing machine in terms of five items: the initial state and the symbol that is read plus the new state, the symbol written, and the motion of the head to the left or right. The machine is specified by a set of quintuples describing exactly what happens for each initial state and symbol read. This set of quintuples may be written out as a binary string. The resulting binary number can now be used to uniquely label the machine by a one-to-one correspondence with the set of natural numbers. In principle then we can now make a list with a natural number specifying each Turing machine and the corresponding number the machine computes. The resulting infinite list now includes every number that is computable! Turing now made use of Cantor’s diagonal slash method to add one to each of the computable numbers on the diagonal as we did in the preceding text to generate a new real number. In this way Turing showed that there are real numbers that are noncomputable. The details of Turing’s proof are a bit more complicated, but this is the basic argument that convinced Turing that the answer to Hilbert’s third question was “no.”

Universality and the Church-Turing thesis

Before we return to the *Entscheidungsproblem*, we need to look at another marvelously original idea in Turing’s paper – the Universal Turing Machine. This is a Turing machine that can do anything that any specific, special-purpose Turing machine can do, albeit more slowly and less efficiently. Suppose we have a specific Turing machine **T** that acts on a tape **t** to produce its result. What Turing showed was that it was possible to construct another Turing machine **U** that, if we give it as input the specification of **T** and the tape **t**, will output the result that machine **T** would have produced acting on tape **t**. The behavior of

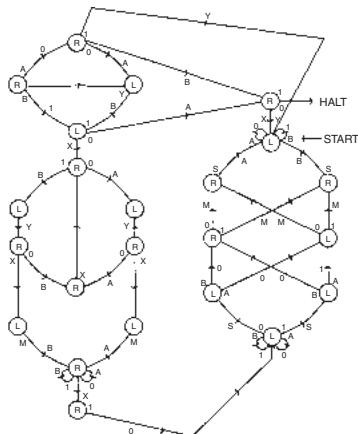


Fig. 6.5. A representation of a Universal Turing Machine due to Marvin Minsky.

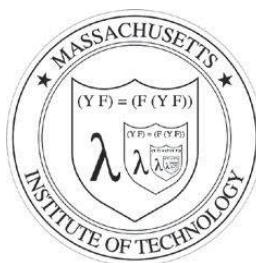


Fig. 6.6. The “Knights of the Lambda Calculus,” the unofficial badge of LISP programmers at MIT.

U is simple to describe but complicated to write down in detail. The Universal Turing Machine **U** must imitate **T** step-by-step, keeping a record of the state of **T**'s tape at each stage. By examining its simulated input tape **t**, the machine can see what **T** would read at any given stage. Then, by looking at the description it has of **T**, **U** can find out what **T** is supposed to do next. This is essentially just what we would do when using a list of quintuples and a tape to figure out what a Turing machine does. Turing's universal machine **U** is just a slower version of us!

Turing went into great detail to prove the existence of a Universal Turing Machine and, as you can see in [Figure 6.5](#), the resulting state transition diagram for **U** is much more complicated than that for our simple Parity Counter ([Fig. 6.10](#)) that we describe in detail later. This example, from MIT computer scientist Marvin Minsky, makes use of eight symbols and twenty-three states. Most digital computers built today are effectively universal computers. With the right program, enough time, and enough memory, any universal computer can simulate any other computer.

If we ignore the slowness and inefficiency of using a Turing machine, we can ask this fundamental question: what problems can be solved by such a machine? The answer is very surprising. Everything that is algorithmically computable is computable by a Turing machine. Why should we believe this? At around the same time as Turing was devising his ingenious machines, Alonzo Church ([B.6.7](#)), a U.S. mathematician based in Princeton, New Jersey, had defined a formalism of logic and propositions that he called *lambda calculus* ([Fig. 6.6](#)). Church argued that any “effectively calculable” problem corresponded to a lambda calculus expression. He had also been able to use his formalism to show that the problem of deciding whether one string of symbols could be converted into another string was unsolvable, in the sense that there was no lambda expression that could do this. In this way Church had been able to show that Hilbert's third question, the *Entscheidungsproblem*, was also unsolvable. Although coming at the problem from very different perspectives, Turing and Church had both proved the problem's insolvability.

The statement – that anything effectively computable is computable by a Turing machine – is known as the *Church-Turing thesis*. It is called a thesis rather than a theorem because it involves the informal concept of effective computability. The thesis equates the mathematically precise statement, “computable by a Turing machine,” with the informal, intuitive idea of a problem being solvable by some algorithm on any machine whatsoever. It applies to all computable problems written in any programming language on any computer!

The Church-Turing thesis is only a thesis but the majority of computer scientists accept its validity because many people besides Church and Turing have arrived at an equivalent result. At about the same time as Church and Turing's work, mathematicians Stephen Kleene and Emil Post devised alternative formalisms that led to similar notions of computability. Many others have looked at variants of the simple Turing machine such as machines with multiple tapes or machines with two-dimensional tapes. None of these machines can solve problems that cannot be solved by the basic Turing machine.



B.6.7. Alonzo Church (1903–95) was very supportive of Turing's ideas and he was first to use the term *Turing machine*. This, along with the Church-Turing thesis on computability, is now one of the cornerstones of computer science.

The halting problem and the *Entscheidungsproblem*

Using Turing's universal machine, it is possible to prove that there is no way to tell, in general, whether the execution of a given program will terminate on any given input. If we have a program to calculate the square of x we can be confident that, after we input x into the machine, we will be able to read x^2 on the tape when the machine halts. Termination is not always so obvious. Suppose we have a program that takes a number as input and either divides it by two, if the number is even, or triples it and adds one, if it is odd. The program then takes this new number as input and repeats the process. When the output is one, the program halts. Can we be sure that this will happen? This is known as the $3x + 1$ problem – what computer scientist David Harel calls the “simplest-to-describe open problem in mathematics.”⁷ If we try this with the starting value of $x = 7$ we get the sequence 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1 and the sequence terminates. With other values of x we find that the program sometimes terminates but for some values of x it just keeps on generating numbers with no repeating pattern until we decide we have seen enough. This is one specific instance of the “halting problem” – a program for which we cannot determine whether it terminates or not. More generally, the halting problem is concerned with the termination of any program for any input.

How can this result be proved? If we have a Turing machine T that calculates some function F , can we find a computable function that predicts whether or not the machine T will halt or not? If there is such a function, we know that it too must be describable by another Turing machine. This concept, of Turing machines telling us about other Turing machines, is a very powerful tool. This device can be used to prove that the halting problem is noncomputable. The trick is to assume the existence of a machine that can predict whether a program halts and then show that this leads to a contradiction, meaning that the original assumption that such a machine exists is incorrect.

We begin our sketch of a proof by supposing we have a machine D that takes as input a tape that contains a description d_T of the machine T – these are just the quintuples that define T – as well as T 's input tape t . Machine D is required to tell us whether T will halt or not and then come to a halt (Fig. 6.7.a). We now introduce another machine Z which takes the machine description d_T and uses this as the input tape for the machine D . This machine Z reacts to the output from D in the following way:

If T halts (D says “yes”), then Z does not halt.

If T does not halt (D says “no”), then Z halts.

We can arrange this to happen by introducing two new states in the “yes” branch. The machine now oscillates between them indefinitely and this prevents our Z machine from halting if D halts (says “yes”) as in Figure 6.7.b. Now we arrive at the crux of the argument. We get Z to operate on itself by taking as input the quintuples d_Z that define the Z machine and substitute Z for T in the above argument. We find:

Z applied to d_Z halts if and only if Z applied to d_Z does not halt.

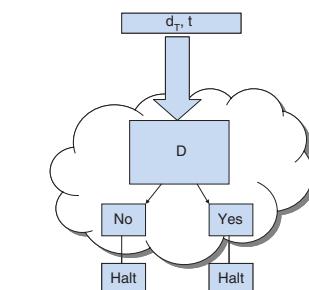


Fig. 6.7a. A hypothetical Turing machine for the halting problem.

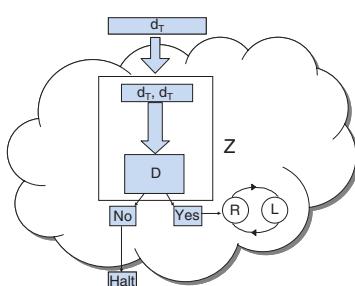


Fig. 6.7b. A paradoxical Turing machine to demonstrate the halting problem.

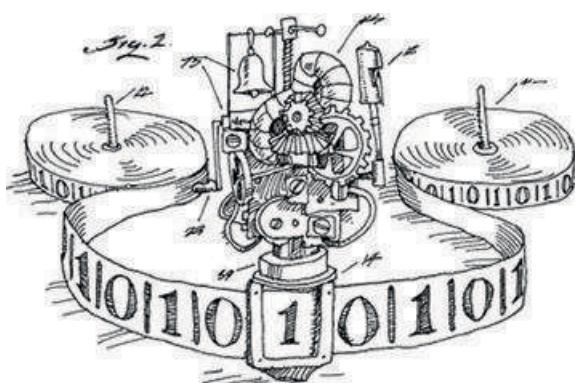
The origin of this contradiction can be traced back to our assumption that machine **D** exists. Therefore no such machine can exist and this shows that the halting problem is not decidable. Using such techniques, Turing was able to demonstrate an unsolvable problem: in so doing, he had shown that Hilbert's *Entscheidungsproblem* has no solution.

The halting problem has a number of important implications. In writing programs we would naturally like to be able to check whether our program actually does do what it is supposed to do. This turns out to be a decision problem. We need to input a description of the algorithmic problem and the text of our program implementing an algorithm that we think solves the problem. We want a "yes" if, for all of the legal inputs for the problem, our algorithm will terminate and give the correct solution; and we want a "no" if there is any input for which our program fails to terminate or gives the wrong result. Now we know about the halting problem we can see immediately that such an automatic verifier is not possible. However, although we cannot guarantee that our program will halt for all inputs, it is still possible to produce formal verification tools that can deliver useful results most of the time!

In another application, Fred Cohen, in 1986 in his doctoral thesis for the University of Southern California, showed that the problem of detecting the presence of a computer virus was an instance of the halting problem. Unfortunately this means that the general problem of identifying a virus cannot be solved. We will have more to say about computer viruses in a later chapter.

Key concepts

- Hilbert's *Entscheidungsproblem* (decision problem)
- Turing machines
- Natural, integer, and rational numbers
- Irrational numbers and effective procedures
- Computable and noncomputable numbers
- Universal Turing Machines
- Church-Turing thesis
- The halting problem



More on Turing machines

Turing's super-typewriter

According to Andrew Hodges, as a boy, Turing dreamed of inventing ways of improving typewriters, and this might have provided him with the starting point for his later ideas about computation. In any case, the way that typewriters manipulate symbols serves as a good introduction to Turing machines. A typewriter is mechanical in that its response to any action of the operator is built in. However, the specific response will depend on whether it was set to type lowercase or uppercase letters; this is the configuration – or state – of the machine. Turing machines generalize this idea to include a larger but still finite number of possible states. A typewriter keyboard contains only a finite number of symbols – the letters of the alphabet and the numbers 0 to 9, plus a few special symbols. Similarly, Turing assumed that his machine was only allowed a finite number of possible operations. Together with the description of the allowed states, this allowed him to write a complete description of the behavior of his machine. The other relevant feature of the typewriter is that the typing point – the point where the typewriter's "head" strikes the paper – can move relative to the page. Turing incorporated this feature – albeit with symbols written on a tape rather than on a page of paper – into his idea for a primitive computing machine.

The typewriter analogy is limited in that a typewriter can only write symbols on a page when they are selected by a human operator, who also decides when to change configuration and where to type the symbol on the page. Turing wanted a much more general kind of machine to manipulate symbols. In addition to writing, Turing wanted his machine to be able to "scan" (that is, read) a symbol on the tape as well as to write or erase a symbol. Such a "super-typewriter" would retain the property of a typewriter in having a finite number of states and an exactly determined behavior for each operation. In addition, unlike in our typewriter analogy, instead of human-operated machines, Turing was interested in investigating what he called *automatic* machines, for which no human intervention would be necessary.

Turing machines in detail

Let us look in more detail at how we can define a Turing machine to do a particular job. We shall label the various possible states of the machine by the symbol Q_i and any particular state "i" as the state Q_i . Similarly, we will label the entries on the tape by the symbol S_j and a particular symbol "j" as the symbol S_j . When we start, only a finite part of the tape has any writing on it: either side of this region, the tape is blank. We start the machine to the left of the writing on the tape at time T. It then proceeds to march along, step-by-step, in uniform time steps, as if following the ticks of a clock. What the state of the machine and the tape is at step $T + 1$ will then be determined by three functions, each of which will depend on the initial state Q_i at step T and the symbol S_j the head has just read. These three functions define what its new state, Q_j will be; what symbol, S_j , it has written on the tape in the original box; and what was the direction, D, of its subsequent motion after writing the new symbol. In mathematical notation, we can write this behavior in terms of three functions – F, G, and D, each depending on the initial Q_i and S_j :

$$Q_j = F(Q_i, S_j)$$

$$S_j = G(Q_i, S_j)$$

$$D_j = D(Q_i, S_j)$$

The Turing machine is fully defined by these three functions, which can be written out as a table of *quintuples*. This is just a fancy name for the set of these two variables and three functions we have defined: Q_i and S_i at time T and Q_j , S_j , and D_j at time $T + 1$. All we have to do now is write some data on the tape and start the machine at the right position. The machine will then calculate away and print out the result of its calculation somewhere on the tape for us to look at when the machine has finished. Note that we have to explicitly instruct the machine when to halt. This sounds pretty trivial but we will see later that the issue of whether a machine will halt or not leads to a profound issue in the theory of computation.

Let's try to construct a very simple Turing machine that measures the *parity* of any string of 0s and 1s. The parity of a string is defined to be whether the number of 1s is even or odd. We are given the string 1101101, and begin by writing this binary string as input data on the tape as shown in Figure 6.8, with one symbol in each box. The reading head of the machine starts at the far left of the string, on the first digit. The end of the string is designated by the letter E. On either side of the string there are only zeros on the tape.

Before it has read any symbols, the machine starts off in state Q_0 , corresponding to even parity. If the machine encounters a 0, it stays in the state Q_0 – because the parity has not changed – and then moves one space to the right. If the symbol it reads is a 1, the machine erases this, replaces it with a 0, moves one space to the right and changes to the state Q_1 . This is the state for odd parity. Continuing, if the machine now hits a 0, it stays in the state Q_1 and moves another space to the right. If it hits a 1, it erases it, prints a 0, and moves another cell to the right, changing the state back to Q_0 . The machine continues working along the string in this way, changing state whenever it encounters a 1, and leaving a string of 0s behind. If the machine is in the state Q_0 after it has read the last symbol, the string has even parity; if it is in the state Q_1 , the parity is odd.

How does the machine tell us the parity and the result of its calculation? We need to include a rule that tells the machine what to do when it meets the end symbol E. If it is in state Q_0 and reads E, it erases the E and writes a 0 meaning that the string had even parity. If it is the state Q_1 , it replaces E by a 1 meaning the string had odd parity. In both cases the machines enter a new state Q_H , meaning “halt.” It does not need to move to the right or the left: the answer can be observed by looking at the box on the tape where the machine halted (Fig. 6.9).

To understand what happens, we have painstakingly described the operation of this Parity Counter Turing Machine in words. In practice, it is more economical to summarize the machine's behavior as a table of quintuples. We can summarize this table of quintuples in a diagram (Fig. 6.10). Here we have indicated the states Q_0 and Q_1 – Even and Odd – by the circles and the direction of motion after reading the box, by R, for right which we also write in the circle. The directed arcs, starting with either a 0 or 1 on them, indicate what happens to

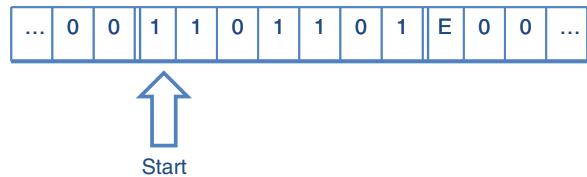


Fig. 6.8. Input tape for the Parity Counter Turing Machine.

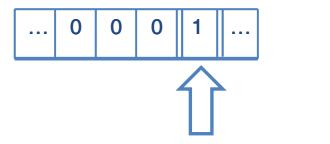


Fig. 6.9. Output tape from the Parity Counter Turing Machine.

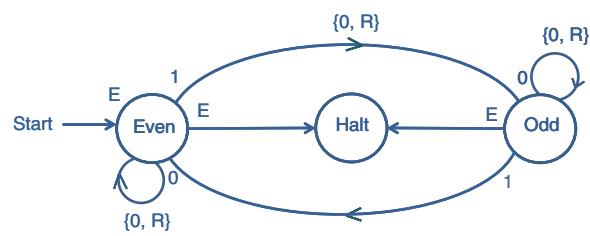


Fig. 6.10. Diagram of a Parity Counter Turing Machine.

the machine if this is the symbol that is read. The symbol on the arc indicates what the head overwrites in the box. Thus reading a 0 from the state $Q_0 = \text{Even}$ just takes us to the same state: reading a 1, takes us to the other circle corresponding to $Q_1 = \text{Odd}$. We have also indicated the start and the halt conditions in the diagram.

Similar diagrams are used to describe the behavior of “Finite State Machines” or FSMs that are often used to summarize the behavior of devices whose actions depend not just on the current input but also on the previous inputs. The FSM has enough memory to store a summary of a finite number of past inputs. A combination lock is an example of an FSM. The lock cannot remember all the numbers dialed into the lock but it remembers enough to know whether the would-be user has entered the correct small sequence of numbers to open the lock. A Turing machine is just an FSM with an infinitely long tape that serves the same function as memory in a computer.

We can now go on to construct Turing machines for adding, multiplying, copying, and so on. To build up more complex machines it is convenient to reuse these simpler machines as components of the complex machine, rather like subroutines in a software program. This greatly simplifies the construction of such machines.

Gödel, von Neumann, Turing, and Church

Gödel and the U.S. Constitution

After Austria's annexation by Germany in 1938, Gödel lost his position at the University of Vienna and was found fit for conscription into the German army. With the outbreak of World War II in 1939, Gödel and his wife set out for Princeton in the United States. Because of the dangers of a North Atlantic crossing, they traveled via the trans-Siberian railway and then by ship across the Pacific. He ran out of money in Japan and had to telegraph to Princeton for a loan (Fig. 6.11).

After the war, Gödel wanted to become an American citizen, and he asked Albert Einstein and economist Oskar Morgenstern to be his witnesses. Of course, Gödel took his preparation for the citizenship hearing very seriously and studied the history of North America, and of Princeton, as well as the U.S. Constitution. At this point, Morgenstern recounts:

[Gödel] rather excitedly told me that in looking at the Constitution, to his distress, he had found some inner contradictions and that he could show how in a perfectly legal manner it would be possible for somebody to become a dictator and set up a Fascist regime, never intended by those who drew up the Constitution.⁸

Einstein and Morgenstern went with Gödel to the citizenship ceremony, and the three of them sat down before the examiner. The examiner first asked Einstein and Morgenstern whether they thought Gödel would make a good citizen, to which they assured him that this would be the case. The examiner then turned to Gödel.

Examiner: Now Mr Gödel, where do you come from?

Gödel: Where do I come from? Austria.

Examiner: What kind of government did you have in Austria?

Gödel: It was a republic, but the constitution was such that it finally was changed into a dictatorship.

Examiner: Oh! This is very bad. This could not happen in this country.

Gödel: Oh yes [it can], I can prove it!⁹

Fortunately, the examiner was a wise man and refrained from following up on Gödel's new inconsistency proof of the U.S. Constitution!

Turing and the conceptual foundation of computers

Although John von Neumann did not refer explicitly to Turing's paper on computability and Turing machines when he wrote the famous "First Draft of a Report on the EDVAC," he was well aware of the importance of Turing's work and even offered him a post as his research assistant at Princeton. The mathematician Stanislaw Ulam, who later worked at Los Alamos on the Manhattan Project, recalled that "von Neumann mentioned to [him] Turing's name several times in 1939 ... concerning mechanical ways to develop formal mathematical systems."¹⁰ Similarly, another physicist who worked at Los Alamos, Stanley Frankel, remembers von Neumann's enthusiasm for Turing's work in 1943 or 1944:

Von Neumann introduced me to that paper and at his urging I studied it with care. Many people have acclaimed von Neumann as the "father of the computer" ... but I am sure that he would never have made that mistake

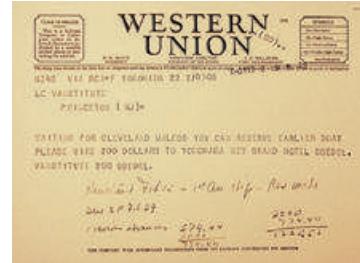


Fig. 6.11. Telegram from Gödel in Yokohama, Japan, to the Institute for Advanced Study in Princeton, requesting \$200 for emergency travel expenses.

himself. He might well be called the midwife, perhaps, but he firmly emphasized to me, and to others I am sure, that the fundamental conception is owing to Turing – insofar as not anticipated by Babbage, Lovelace, and others. In my view von Neumann’s essential role was in making the world aware of these fundamental concepts introduced by Turing and of the development work carried out in the Moore school and elsewhere.¹¹

And in 1946, von Neumann wrote to his friend Norbert Wiener of “the great positive contribution of Turing … one, definite mechanism can be ‘universal.’”¹²

Some of the early pioneers of computers did not recognize that they were, in essence, building a variant of a universal Turing Machine. In 1956 Howard Aiken said:

[If] it should turn out that the basic logics of a machine designed for the numerical solution of differential equations coincide with the logics of a machine intended to make bills for a department store, I would regard this as the most amazing coincidence I have ever encountered.¹³

Alan Turing himself, in contrast with Howard Aiken’s remarks, said in 1950:

This special property of digital computers, that they can mimic any discrete state machine, is described by saying that they are universal machines. The existence of machines with this property has the important consequence that, considerations of speed apart, it is unnecessary to design various new machines to do various computing processes. They can all be done with one digital computer, suitably programmed for each case. It will be seen that as a consequence of this all digital computers are in a sense equivalent.¹⁴

In 1945 Turing produced a report for the construction of his ACE Automatic Computing Engine. Compared to von Neumann’s EDVAC Report, which “is a draft and is unfinished,” the ACE Report “is a complete description of a computer, right down to the logical circuit diagrams.” In contrast to the computer designs based on the EDVAC ideas – which were focused on delivering fast numerical calculations – Turing’s design recognized the full power of digital computers as all-purpose machines, capable of manipulating symbols and playing chess as well as performing numerical operations. To characterize these designs as merely embodying the “stored-program concept” is to underappreciate the breadth of Turing’s vision – of which von Neumann was well aware.

Turing and Church

In April 1936, Turing had just delivered his paper to Max Newman, much to Newman’s surprise. Newman read the paper and realized the significance of Turing’s work. He encouraged Turing to publish the paper in the *Proceedings of the London Mathematical Society*. As Turing was tidying up his paper for publication, in mid-May Newman received a copy of Church’s paper. Since the subject matter of the two papers had much overlap and Church had priority in terms of publication, there was some doubt as to whether Turing’s paper could be published. Newman wrote to the editor of the journal:

I think you know the history of Turing’s paper on Computable numbers. Just as it was reaching its final state an offprint arrived, from Alonzo Church of Princeton, of a paper anticipating Turing’s results to a large extent. I hope it will nevertheless be possible to publish the paper. The methods are to a large extent different, and the result is so important that different treatments of it should be of interest.¹⁵

Fortunately, the editor agreed, and Turing’s paper with his machines was published in the *Proceedings*. Newman also wrote to Church in Princeton:

Dear Professor Church,

31 May 1936

An offprint which you kindly sent me recently of your paper in which you define ‘calculable numbers’, and show that the Entscheidungsproblem for Hilbert logic is insoluble, had a rather painful interest for a young man, A. M. Turing, here, who was just about to send in for publication a paper in which he had used a definition of ‘Computable numbers’ for the same purpose. His treatment – which consists in describing a machine which will grind out any computable sequence – is rather different from yours, but seems to be of great merit, and I think it is of great importance that he should come and work with you next year if that is at all possible. He is sending you the typescript of his paper for your criticisms.... I should mention that Turing’s work is entirely independent: he has been working without any supervision or criticism from anyone. This makes it all the more important that he should come into contact as soon as possible with the leading workers on this line, so that he should not develop into a confirmed solitary.¹⁶

Turing read Church’s paper in the summer and added an appendix to his paper that demonstrated that his definition of ‘computable’ – meaning anything that could be computed by a Turing machine – was equivalent to what Church had called “effectively calculable” – meaning anything that could be described by a formula using the lambda calculus. When Turing’s paper was published in January 1937, Church generously reviewed it very positively in the well-known *Journal of Symbolic Logic*. He also used the description “Turing machine” in print for the first time, writing that “a human calculator, provided with pencil and paper and explicit instructions, can be regarded as a type of Turing machine.”¹⁷

7 Moore's law and the silicon revolution

As I prepared for this event, I began to have serious doubts about my sanity. My calculations were telling me that, contrary to all the current lore in the field, we could **scale** down the technology such that **everything got better**: the circuits got more complex, they ran faster, and they took less power – **WOW!**

Carver Mead¹

Silicon and semiconductors

When we left the early history of computers in [Chapter 2](#), we had seen that logic gates were first implemented using electromechanical relays – as in the Harvard Mark 1 – and then with vacuum tubes – as in the ENIAC and the first commercial computers. These early computers with many thousands of vacuum tubes actually worked much better and more reliably than many engineers had expected. Nevertheless, the hunt was on for a more dependable technology. After World War II, Bell Labs ([Fig. 7.1](#)) initiated a research program to develop solid-state devices as a replacement for vacuum tubes. The focus of the program was not on materials that were metals or insulators but on strange, “in-between” materials called *semiconductors*.

In a solid, it is the flow of electrons that gives rise to electric currents when a voltage is applied. One of the great successes of quantum physics has been in giving us an understanding of the way in which different types of solids – metals, insulators, and semiconductors – conduct electricity. This quantum mechanical understanding of materials has led directly to the present technological revolution, with its accompanying avalanche of stereo systems, color TVs, computers, and mobile phones. A good conductor, such as copper, must have many *conduction electrons* that are able to move and thus constitute a current when a voltage is applied. By contrast, an insulator such as glass or carbon has very few conduction electrons, and little or no current flows when a voltage is applied. Semiconductors are solids that conduct electricity much better than insulators but much worse than metals. The elements germanium and silicon are two examples. The importance of silicon for computer technology is evident in the naming of California’s “Silicon Valley,” home to many of the earliest electronic component manufacturers ([Fig. 7.2](#)).

The properties of a solid depend not only on what element it is made of, but also on the way the atoms or molecules are stacked together. Many solid



Fig. 7.1. An aerial view of AT&T Bell Labs in Holmdel, New Jersey. The building was designed by the architect Eero Saarinen and for forty-four years it was the home of an advanced research laboratory owned successively by Bell Telephone, AT&T, Lucent, and Alcatel.



Fig. 7.2. A Landsat photograph of Silicon Valley and San Francisco Bay. In 1971 journalist Don Hoefler ran a series of articles in *Electronic News* under the title "Silicon Valley USA" and the name caught on.

materials have their constituent atoms arranged in a regular array, like bricks in a wall but in three dimensions. This regular pattern of atoms is called a *crystal lattice*, and substances with such a structure are called *crystalline solids*. Arranging all the atoms in a regular array has a dramatic effect on the allowed energy levels for the atomic electrons. The way to understand the energy levels of such crystalline materials was discovered by a Swiss physicist named Felix Bloch. To find the allowed electron energy levels for any quantum mechanical system, you need to solve the Schrödinger equation – a mathematical formula as fundamental for the behavior of quantum objects as Newton's laws are for classical objects. Solving this equation for an electron in the potential of a positively charged nucleus leads to definite, isolated energy levels. For electrons in a potential corresponding to a regular lattice of positive ions, Bloch found that instead of isolated energy levels, the allowed energy levels merged into several "bands" of allowed energies. The discovery of such *energy band structures* provides the foundation for our understanding of the difference between metals, semiconductors, and insulators. Figure 7.3 shows typical allowed energy band structures for these three types of materials.

In a metal such as copper, the lowest energy band has many unfilled levels and the conduction electrons can move freely into empty levels, gaining energy when a voltage is applied and generating an electric current (Fig. 7.3a). At absolute zero, the coldest possible temperature (-273.15°C), the energy levels in the bands would be filled up one electron at a time, according to the Pauli Principle to give the minimum energy state (see the quantum theory primer at the end of this chapter for more details). At room temperatures, the lattice ions have some

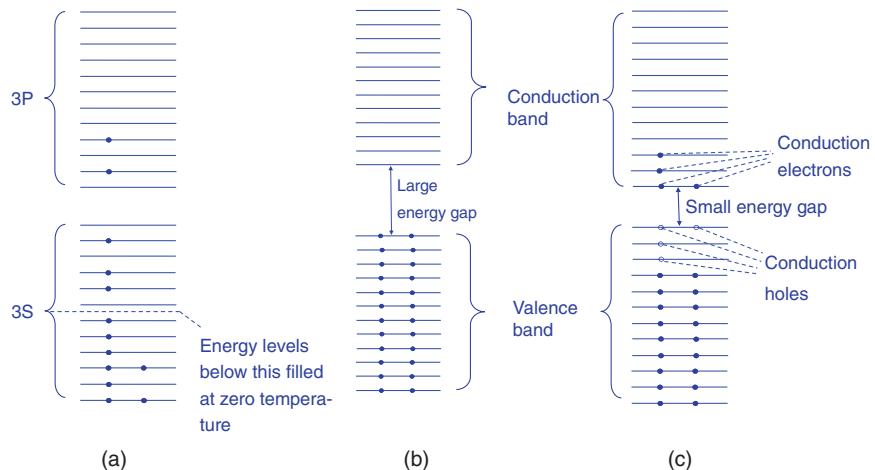


Fig. 7.3. Band structures of metals, insulators, and semiconductors. (a) Band structure of a typical metal like sodium. There are many unfilled energy levels in the "3S" valence band for the conduction electrons to occupy. At normal temperatures, only a few electrons will be excited into the almost empty "3P" band. (b) In an insulator, the valence band is full and the gap between the valence and conduction bands is too large for any significant number of electrons to jump across the gap with normal thermal energy distributions. As a result, an insulator conducts electricity very poorly, if at all. (c) In a semiconductor, the valence band is almost full but there is only a small energy gap to the mostly empty energy levels in the conduction band. At normal temperatures, some of the electrons have enough thermal energy to be able to jump across this energy gap.

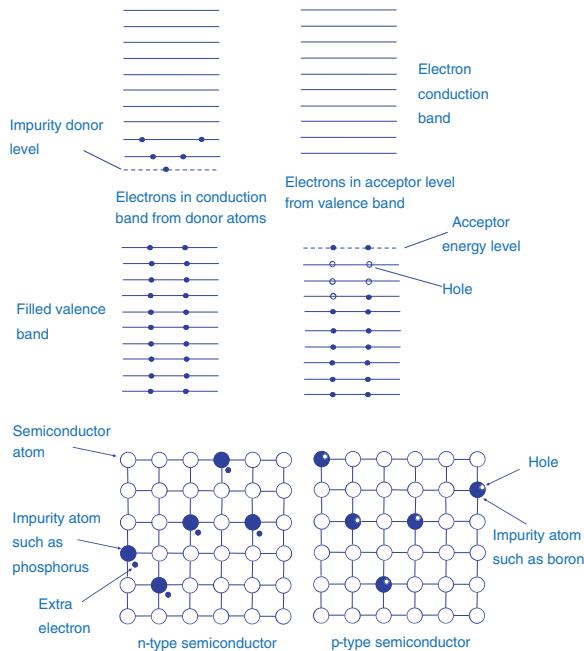
thermal kinetic energy corresponding to vibrations about their positions in the crystal lattice. As the conduction electrons move through the metal they can gain or lose energy in collisions with the lattice ions and with each other. Thus, instead of the conduction electrons filling up only the lowest energy levels in the band, some electrons will be *thermally excited* to higher levels in the band and even to higher bands. This has the effect of leaving some empty energy levels in the bottom of the lowest band.

For an insulator like carbon, the lowest energy band is full and there is a large energy gap to the next band (Fig. 7.3b). In this case, almost no electrons are able to gain enough energy from collisions to jump into the empty, higher energy band. When a voltage is applied to the material, there are therefore no empty levels close by for the electrons to be able to move to and gain energy, so the material acts as an insulator. The energy bands in a semiconductor are shown in Figure 7.3c. These materials have a similar band structure to an insulator with the bottom band filled, but the energy gap to the next band of energy levels is much smaller. At ordinary temperatures, some electrons are excited by thermal collisions into the upper conduction band. When a voltage is applied, the electrons in the upper band have plenty of empty states to move to and allow the electrons to gain energy. There will also be some empty states in the lower band that allow conduction. Thus semiconductors will conduct currents fairly easily, but their conductivity will depend strongly on temperature, in contrast to metals and insulators.

Two Nobel Prizes: The transistor and the integrated circuit

Pure semiconductors are not in themselves of great practical importance. In metals almost every atom contributes one or more conduction electrons but in semiconductors only one atom in about a thousand million contributes an electron to conduct electricity. This apparent drawback has the great advantage that the conduction properties of semiconductors can be easily modified by introducing tiny amounts of additives called *impurity atoms* – at the level of around one atom in a million. Germanium and silicon both have four *valence electrons*, electrons in the atom's outermost shell that can be easily transferred to or shared with other atoms. The valence electrons fill up most of the states in the *valence band*, which lies below the almost-empty *conduction band*. If we introduce an impurity such as phosphorous, which has five valence electrons, into the pure semiconductor only four of these electrons are needed to maintain the crystal lattice structure. As a result there will be an electron left over that can easily be detached from the phosphorous atom and contribute to the conductivity. Similarly, if we introduce an impurity atom such as boron, with only three valence electrons, there will be one electron missing in the bonds that hold the lattice together. The missing electron creates a site that can capture electrons from filled states in the valence band, leaving empty states and allowing some conduction. These two situations are represented on the energy level diagram shown in Figure 7.4. The process of adding impurity atoms is called *doping*. Semiconductors that have been doped with phosphorus are called *n-type semiconductors*. The phosphorus atoms give rise to electron *donor states* just

Fig. 7.4. Semiconductors doped with impurity atoms. (a) n-type semiconductor in which the impurity atoms have an extra electron. This results in the effective energy-level diagram shown here with a “donor level” just below the conduction band. (b) p-type semiconductor doped with impurity atoms with one fewer electron, resulting in electron “holes.” The equivalent energy-level diagram has an empty “acceptor level” just above the valence band.



B.7.1. Russell Shoemaker Ohl (1898–1987) was a researcher investigating the behavior of semiconductors at AT&T's Bell Labs in Holmdel, New Jersey. In 1939, Ohl discovered the “p-n junction” by which he was able to manipulate current flows. He also recognized the importance of using exceptionally pure semiconductor crystals to make repeatable and usable semiconductor diodes. His work with these devices led him to develop and patent the first silicon solar cells.

below the conduction band, and these electrons need only gain a small amount of energy to jump into the conduction band. Semiconductors doped with boron are called *p-type semiconductors*. The boron atoms give rise to electron *acceptor* states just above the nearly full valence band and at room temperatures electrons are readily excited into these levels. Compared to an undoped semiconductor, the boron impurity site is missing a negatively charged electron. This is equivalent to the p-type semiconductor having a positive charge compared to the undoped material. Conductivity in the nearly full valence band is possible because electrons can move into the unoccupied “hole” states. In a p-type semiconductor, instead of thinking of a negatively charged electron moving in response to a voltage, we can equally well think of a positively charged *hole* moving in the opposite direction. Because moving a negative charge to the left has the effect of increasing the charge on the right, we can alternatively think of the current as a flow of positively charged holes moving to the right.

Why is all this useful? Russell Ohl (**B.7.1**) at Bell Labs had discovered that p- and n-type semiconductors could be put together to form interesting semiconductor devices. The simplest device is the *p-n junction diode*, which prevents current from flowing in one direction but not the other. This p-n junction device is able to convert an alternating current into a unidirectional current – a property called *rectification*. The development of the p-n junction diode was the first step toward the invention of the transistor, a semiconductor device that could be used either to amplify a signal or to switch a circuit on or off. John Bardeen, Walter Brattain, and William Shockley (**B.7.2**) were awarded the 1956 Nobel Prize for physics for their invention of the transistor. The transistor was not discovered by accident – it was the culmination of an extensive research program at Bell Labs. As Bardeen later said in his Nobel Prize lecture: “The

Fig. 7.5. The first transistors: (a) replica of the point-contact transistor invented by John Bardeen and Walter Brattain. The wedge of semiconductor that forms the base is about three centimeters on each side. (b) William Shockley's junction transistor consisted of a thin layer of n-type semiconductor sandwiched between two thicker regions of p-type material.

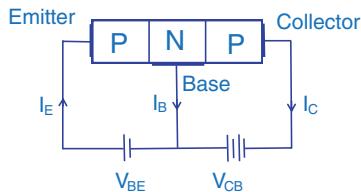
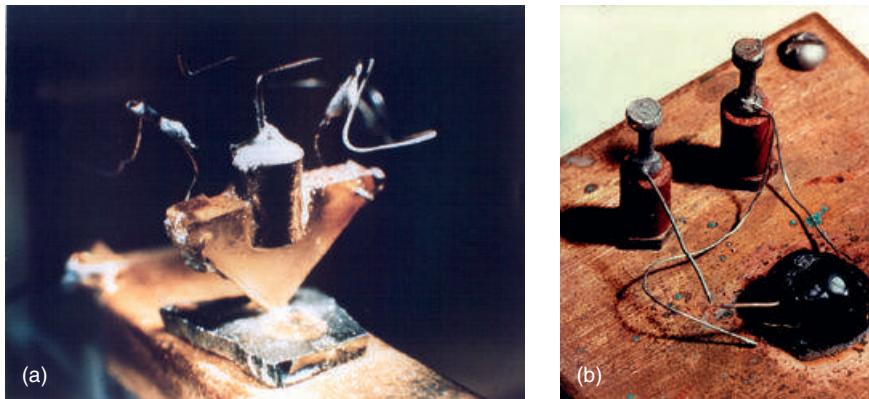


Fig. 7.6. Schematic of a PNP transistor illustrating the direction of currents (I) and voltages (V). The current to the emitter is the sum of the currents of the base and collector ($I_E = I_B + I_C$).

general aim of the program was to obtain as complete an understanding as possible of semiconductor phenomena, not in empirical terms, but on the basis of atomic theory.”² A replica of Bardeen and Brattain’s first transistor, called a *point-contact transistor*, is shown in Figure 7.5a. The two scientists first succeeded in observing amplification of a signal by this device on 24 December 1947. This discovery was followed in 1951 by Shockley’s *p-n-p junction transistor* (Fig. 7.4b). This device turned out to be much more reliable and easier to manufacture than the point-contact transistor.

A junction transistor consists of a thin layer of n-type semiconductor sandwiched between two thicker regions of p-type material (Fig. 7.6). A transistor has three *electrodes*, conductors that can emit or collect electrons or holes, or that can be used to control the movement of the current through the device. Thus the current flowing in the electrode called the *collector* is controlled by a small current applied to the electrode called the *base*. In a p-n-p transistor, a large current through the high-resistance collector-base p-n junction can be controlled by a small current through the low-resistance base-emitter n-p junction (see Fig. 7.6). This action can be understood by a detailed consideration of the energy levels and the electron and hole currents across the two p-n junctions. The word *transistor* refers to this effect and comes from combining the two words *transfer* and *resistor*. The first commercial application of transistors was in hearing aids, followed soon after by the first portable transistor radio produced in 1955 by a company called Regency in Indianapolis. However, transistors also proved to be ideal for implementing the “on-off” binary logic of computers. Their speed and reliability, together with a large number of incredible engineering advances, have made them the basic ingredient of modern microelectronics.

One of the most important of the engineering advances leading to the development of the modern electronics industry was first envisioned by a British engineer named Geoffrey Dummer (B.7.3). He worked at the Telecommunications Research Establishment (later the Royal Radar Research Establishment), a research facility in Malvern, England. Dummer was an expert on the reliability of electronic components and was concerned about the performance of radar equipment under extreme conditions. He realized that it was both inefficient



B.7.2. The three inventors of the transistor, from left to right, John Bardeen (1908–91), William Shockley (1910–89), and Walter Brattain (1902–87). They were awarded the 1956 Nobel Prize for Physics. Bardeen went on to win a share of a second Nobel Prize for Physics for his work on the theory of superconductivity.



Fig. 7.7. Jack Kilby's first IC. Instead of making the components of the electronic circuit separately, Kilby incorporated a junction transistor, a capacitor, and resistances in the same piece of germanium. The device is 1/16 by 7/16 inches or 1.6 × 11.1 mm.



B.7.3. Geoffrey Dummer (1909–2002) was a researcher at the Royal Radar Research Establishment in Malvern, England. In the IC manufacturing world he was known as the “The Prophet of the Integrated Circuit.” His vision for an IC was motivated by a desire to make electronic components more reliable.

and unnecessary to manufacture each of the components of an electronic circuit in separate pieces. If all these devices could be contained in the same piece of semiconductor, the circuit would be much smaller and more reliable. In May 1952, Dummer wrote:

With the advent of the transistor and the work in semiconductors generally, it seems now possible to envisage electronic equipment in a solid block with no connecting wires. The block may consist of layers of insulating, conducting, rectifying and amplifying materials, the electrical functions being connected directly by cutting out areas of the various layers.³

Dummer’s description was an amazingly accurate vision of a modern *integrated circuit*, or IC, a circuit etched or imprinted on a slice of semiconductor. But there was a long way to go to make such an IC an engineering reality.

The vital breakthrough was made in the summer of 1958 by an American electrical engineer named Jack Kilby. In the early 1950s, Kilby had worked on printed circuits, transistors, and the miniaturization of electronics, which were of great interest to the U.S. military. He then joined Texas Instruments, a semiconductor manufacturing company, to work “in the general area of microminiaturization.”⁴ Kilby arrived in the summer, just before most of the employees took their summer vacations:

Since I had just started and had no vacation time, I was left pretty much in a deserted plant; so I began to think ... I began to cast around for alternatives [to making circuits out of individual components] – and the monolithic [or solid circuit] concept really occurred to me during that two-week vacation period. I had it all written up by the time Willis [engineer Willis Adcock, who helped develop the silicon transistor] got back, and I was able to show him the sketches that pretty well outlined the idea – and the process sequence showing how to go about building it.⁵

By September 1958, Kilby had built the first working IC, all from a single piece of germanium (Fig. 7.7). The device was an oscillator (a circuit that generates a regular signal), containing a single junction transistor; a capacitor to store electrical energy; and several resistors to limit the electrical current – all made from a single piece of semiconductor. Kilby wired together the different components of the circuit by soldering tiny wires to his device. His version of the IC was limited by the difficulty of physically wiring up many components, but it was still a major breakthrough. Kilby was rapidly converted from electrical engineer to



B.7.4. Jack Kilby (1923–2005) and Robert Noyce (1927–90) display their medals at the first Draper Prize award in 1989. The citation for their award was “for their independent development of the monolithic integrated circuit.” Kilby went on to gain a share of the 2000 Nobel Prize for Physics. In his Nobel Lecture he made an explicit acknowledgment of Noyce’s contribution: “I would like to mention another right person at the right time, namely Robert Noyce, a contemporary of mine who worked at Fairchild Semiconductor. While Robert and I followed our own paths, we worked hard together to achieve commercial acceptance for integrated circuits. If he were still living, I have no doubt we would have shared this prize.”⁶¹



Fig. 7.8. Memorial sign for the site of the original Shockley Semiconductor Laboratory. In later life, Shockley became a controversial figure for his views on race and genetics. Despite this controversy it seems regrettable that an earlier plaque with Shockley's name on it has been replaced with this more "politically correct" version.

physicist by the Nobel Prize committee when he was awarded the Nobel Prize for physics in 2000 for his invention of the IC (B.7.4).

The beginnings of Silicon Valley

After his invention of the junction transistor, Shockley had a falling out with both Bardeen and Brattain. As a result, Bardeen left Bell Labs in 1951 to be a professor at the University of Illinois at Urbana-Champaign. In 1972, Bardeen won his second Nobel Prize for his role in developing the “BCS” (Bardeen, Cooper, and Schrieffer) theory of superconductivity, the only physicist to have been awarded two Nobel Prizes in physics. Shockley took leave of absence from Bell Labs in 1953 and, with the help of a Caltech friend, Arnold Beckman, set up the Shockley Semiconductor Laboratory, a division of Beckman Instruments in Mountain View, California, in 1955 (Fig. 7.8). Mountain View was near Palo Alto, Shockley’s hometown and the location of Stanford University. Three of the first recruits to Shockley’s company were physicists Robert Noyce and Jean Hoerni and chemist Gordon Moore. Sadly, Shockley was a terrible people manager and eventually alienated most of his employees. By the summer of 1957, Noyce, Hoerni, Moore, and five others – the “Traitorous Eight” – decided to leave Shockley and set up a company by themselves (B.7.5). With financing from the Fairchild Camera and Instrument Corporation, they formed Fairchild Semiconductor. While their building was under construction, their temporary workplace was a large garage in Palo Alto – now a tradition for Silicon Valley start-ups!

Two key innovations were needed to make the mass production of powerful and robust ICs a reality. At the time, the state of the art in transistors was the silicon *mesa transistor*, which had a tiny round plateau or “mesa” of silicon set on top of a base, also made of silicon (Fig. 7.9). Because the contacts on the mesa structure were exposed, these transistors were easily contaminated or damaged. Soon after Fairchild Semiconductor was established, the Swiss physicist Jean Hoerni (B.7.6) came up with a brilliant innovation, which now underpins all modern ICs. His idea was the *planar transistor*, where the mesa is now fully embedded into the silicon wafer, resulting in a completely flat transistor (Fig. 7.10). Hoerni also coated his device with a layer of silicon dioxide, which insulated and protected the transistor’s contacts. The remaining obstacle to mass production was a method to electrically isolate the components in the silicon. This was solved in late 1958 by the Czech-born physicist Kurt Lehovec, who worked for the Sprague Electric Company in Massachusetts. He had heard of Kilby’s IC and realized the importance of isolating the different components in the silicon. His solution was very simple: he proposed inserting back-to-back p-n junctions, or diodes, between the transistors in the silicon so that no current could flow in either direction. All these ideas came together in January 1959 when Noyce (B.7.7) developed a process for manufacturing ICs using Hoerni’s planar transistors and Lehovec’s p-n junctions. As Noyce said later:

When this [the planar process] was accomplished, we had a silicon surface covered with one of the best insulators known to man, so you could etch holes through to make contact with the underlying silicon. Obviously, then,

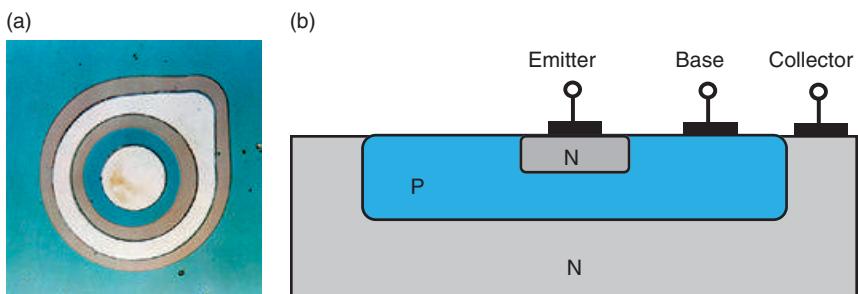


B.7.5. A photograph of the Traitorous Eight – the founders of Fairchild Semiconductor who left Shockley’s original Silicon Valley start-up. From left to right, they are Gordon Moore, Sheldon Roberts, Eugene Kleiner, Robert Noyce, Victor Gingrich, Julius Blank, Jean Hoerni, and Jay Last.

Fig. 7.9. In geology, a mesa is a flat-topped mountain, typically found in the U.S. Southwest as in this example in Monument Valley. In the same way, a semiconductor mesa transistor also rises above the surrounding semiconductor base with a height typically less than one micron.



Fig. 7.10. (a) A view of Hoerni's planar transistor under the microscope. (b) A diagram showing the bowls of p-type semiconductor and n-type semiconductor together with connections for the emitter, base, and collector. The entire surface would be coated with silicon dioxide.



you had a whole bunch of transistors embedded in an insulating surface, and the next thing was that, instead of cutting them apart physically, you cut them apart electrically, added the other components you needed for circuits, and finally the interconnection wiring.⁶

There were several techniques, but the main one was, basically, to build back to back diodes [p-n junctions] into the silicon between any two transistors so that no current could flow between the two in either direction. The other element you needed was a resistor, and it was relatively simple to make a diode-isolated piece of silicon that acts as a resistor. You now had resistors and transistors, and could start building logic circuits, which you could interconnect by evaporating metal on top of the insulating layer. So it was a progressive buildup of bits and pieces of the technology to make the whole thing possible.⁷

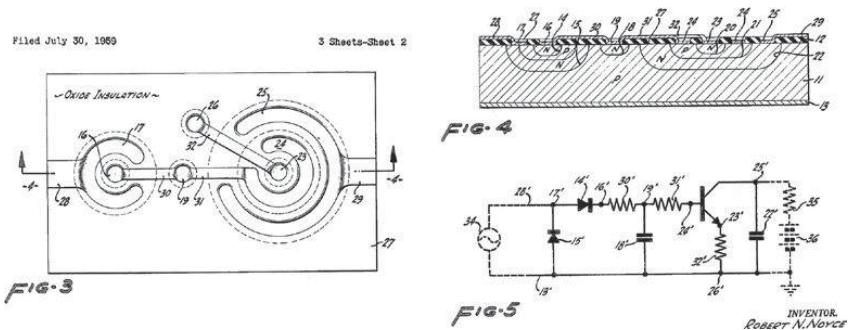
Noyce's patent for his version of the IC was filed in July 1959 (Fig. 7.11). His assembly of the key component technologies produced a design for a circuit that could be mass produced. Because ICs were made from tiny chips of silicon, they became known as *chips* or *microchips*. Fairchild started marketing a whole family of logic chips – the decision-making units of computers – in 1962.

Also in 1962, a new type of transistor debuted that could be more easily incorporated in mass-produced chips. This was the MOSFET, the *metal-oxide-semiconductor field effect transistor*. It was first successfully produced by John Atalla and Dawon Khang of Bell Labs in 1959. Bell Labs did not pursue the



B.7.6. Jean Hoerni (1927-94) was the inventor of the planar process that revolutionized chip production.

Fig. 7.11. Diagrams from Robert Noyce's patent for ICs. His design used Jean Hoerni's planar transistors with Kurt Lehovec's p-n junctions to isolate the different electrical components.

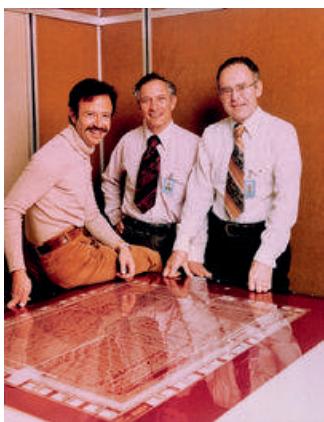


technology but Khang commented on its potential in a 1961 memo because of its “ease of fabrication and the possibility of application in integrated circuits.”⁸ It was left to two young engineers, Steven Hofstein and Frederic Heiman, at RCA Corporation’s research laboratory in New Jersey to build an experimental IC using sixteen metal-oxide-semiconductor (MOS) transistors. Because of their small size and low power consumption, more than 99 percent of the microchips produced today use MOSFET transistors. Both p-type MOSFETs and n-type MOSFETs are employed in the dominant technology for constructing ICs, a method known as complementary metal-oxide-semiconductor (CMOS) technology.

Chip development continued apace, with ever-increasing miniaturization and complexity. By 1967, chips were being produced that incorporated thousands of transistors. Although the initial steps toward IC development had little to do with funding from the U.S. military sector, the U.S. military and the aerospace community played a key role in improving the quality of chips and developing better techniques for their mass production. In the early 1960s, at the height of the Cold War, the U.S. Air Force needed to expand its Minuteman ballistic missile program (Fig. 7.12). It looked to ICs to replace the discrete electronic components and increase the computing power (Fig. 7.13). The air force wanted to produce missiles at a rate of “around six to seven missiles a week,”⁹ and it ordered more than four thousand ICs per week from Texas Instruments, Westinghouse Electric Corporation, and RCA. The air force’s insistence on more reliable components also forced suppliers to introduce “clean rooms” – facilities with little dust and other pollutants, adapted from those developed at Sandia National Laboratories in New Mexico for the assembly of atomic weapons.

This expansion of mass production of ICs drove down the costs for later consumer applications. Together the U.S. Air Force and Navy programs accounted for the entire \$4 million IC market in 1962; by 1968, the U.S. government accounted for only 40 percent of a \$300 million IC market. From 1962 to 1968 the average price of an IC dropped from more than \$50 per microchip to about \$2.

In 1959, Fairchild Camera and Instrument bought out the eight founders of Fairchild Semiconductor. The parent company then introduced a more rigid management style that triggered an exodus of the founders and other talent. From its beginning with Shockley’s first semiconductor company in Mountain View, the diaspora from Fairchild Semiconductor led to the establishment of



B.7.7. A photograph of Andy Grove, Robert Noyce, and Gordon Moore who were responsible for making Intel such a successful chip manufacturer. Grove, a Hungarian-born engineer and businessman, was one of the evangelists behind Intel’s relentless drive to pack more and more transistors on a chip. Grove’s book *Only the Paranoid Survive* has become a classic in business management. Noyce’s nickname was the “Mayor of Silicon Valley.”



Fig. 7.12. Minuteman missile silo, South Dakota, United States.

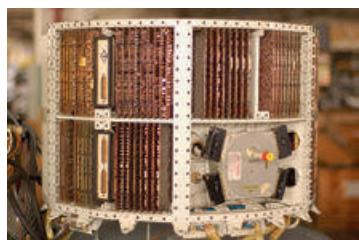


Fig. 7.13. Photograph of the Minuteman I missile guidance computer assembled from discrete electronic components: transistors (1,521), diodes (6,282), resistors (504), and capacitors (1,116). The computer was specially designed to survive and function in extreme conditions. In the Minuteman II, ICs manufactured by Texas Instruments replaced multiple transistor boards. The Minuteman II computer used about two thousand ICs.

more than fifty IC companies in and around San Jose, California. Noyce and Moore were the last of the founders to leave Fairchild. In 1968, they set up a new company with the intention of specializing in memory chips. The name of their new company was Intel Corporation, short for *integrated electronics*. At that time, the best random access memory (RAM) chip had a capacity of sixty-four bits, which was not sufficient to supplant magnetic core memory in computers. RAM refers to data storage that allows information to be accessed in any order, unlike storage on a magnetic tape, which can only access information in a linear fashion by moving along the tape. By storing frequently used or active files in RAM, the computer can access the data much faster from RAM than it can retrieve information from a hard disk – and very much faster and much more flexibly than from magnetic tape.

In 1968, Bob Dennard (B.7.8) from IBM patented a one-transistor design for *dynamic RAM*, or DRAM. In his design, a single bit of information can be stored in a memory cell consisting of one transistor and a tiny capacitor. This innovation simplified the design of memory chips and permitted a significant increase in memory capacity. Dynamic memory is so-called because the stored charge leaks slowly away and the memory cell needs to be regularly refreshed to maintain its contents. It is possible to design *static RAM* (SRAM) chips that do not need refreshing but this type of RAM needs more transistors to implement and is therefore more expensive. In 1970, Fairchild, now a competitor to Moore and Noyce's new company, brought out a 256-bit DRAM memory chip. This chip received much momentum by being chosen as the memory for an ambitious “parallel computer,” the Illiac-IV, under construction at the University of Illinois. A parallel computer has many processing units and a programmer has to orchestrate the work of all of these units to solve a problem. Although the Illiac-IV computer had only limited success – parallel programming is still too hard – the project showed that semiconductor memory was a viable alternative to magnetic cores. By the end of 1970, Intel had responded to Fairchild's 256-bit chip by introducing the 1103, the first 1,024-bit DRAM chip, using a three-transistor design. In 1971 Intel's revenues were about \$9 million; three years later these had almost tripled. The end of magnetic core memories was in sight.

The microprocessor and Moore's law

The first electronic calculator was called the ANITA and was produced in 1961 by the Bell Punch Company in the United Kingdom. It used discrete transistors and was about the size of a typewriter. Later in the 1960s, Texas Instruments built a calculator using ICs. Some were logic circuits for doing the calculations, others were RAM, and still other circuits provided *read-only memory* (ROM) for the operating system and subroutine libraries. The contents of ROM can be accessed and read but cannot be changed. When cheap pocket versions appeared in the 1970s, the traditional slide rule of the engineer rapidly disappeared. At the beginning of NASA's Apollo space program in 1963, the guidance computer of the lunar module was constructed from about five thousand logic chips. On the last Apollo mission in 1975, one astronaut carried a Hewlett Packard HP-65 pocket calculator more powerful than the spacecraft's guidance computer.



B.7.8. A photograph of Robert Dennard from IBM. On the white board behind him is a sketch of his one-transistor DRAM cell. This led to the widespread availability of cheap semiconductor memory chips. His clear statement of the consequences of the physics behind Moore's law is known as *Dennard scaling*.

In the summer of 1969, the Japanese calculator manufacturer Busicom asked Intel to design a set of chips for their new range of *programmable calculators*, which could be programmed much like a computer. The Japanese engineers had a design that involved twelve logic and memory chips, each with several thousand transistors. Ted Hoff, the Intel engineer assigned to the project, thought that this was not a very efficient solution to their problem. Instead, Hoff suggested developing a general-purpose logic chip that, like the central processor of a computer, could be programmed to perform any logical task (B.7.9). Together with some RAM and ROM, and a chip to control input and output, Hoff's solution needed only four chips to be designed instead of the original twelve. Intel engineers Stan Mazor and Frederico Faggin, together with Busicom engineer Masatoshi Shima, implemented the design (Fig. 7.14). This was the first *microprocessor* sold as a component – an IC that can perform all of the functions of a computer's central processing unit (CPU). At first Intel was not sure of the market for its microprocessor (called the 4004) because the company thought that there was too little demand for calculators. As it turned out, any machine that handled and manipulated information or controlled a complex process was a potential market for the microprocessor.

The Intel 4004 microprocessor was launched in 1971. It contained more than two thousand transistors and measured 1/8 inch by 1/16 inch a side. This single chip had about the same computing power as the original ENIAC computer. Intel introduced a more powerful microprocessor called the 8080 in 1974. The 8080 led to a host of new applications and, as we shall see in the next chapter, to the creation of the personal computer. By the early 1980s Intel had more than \$1 billion in sales: twenty years later, the global market for microprocessors was more than \$40 billion. The vast majority of microprocessor chips are used in “embedded” applications – such as in washing machines, cookers, elevators, airbags, cameras, TVs, DVD players, and mobile phones. Automobiles and planes are increasingly reliant on microprocessors, as are the many of the infrastructure systems vital to the functioning of a modern city.



B.7.9. The three inventors of Intel's first microprocessor, Ted Hoff, Stan Mazor, and Frederico Faggin. Hoff was Intel employee number 12; he and colleagues Mazor and Faggin were awarded the 2010 U.S. National Medal for Technology and Innovation. Faggin credits Masatoshi Shima, an engineer from the Japanese company Busicom, for help with the detailed design work for the 4004.

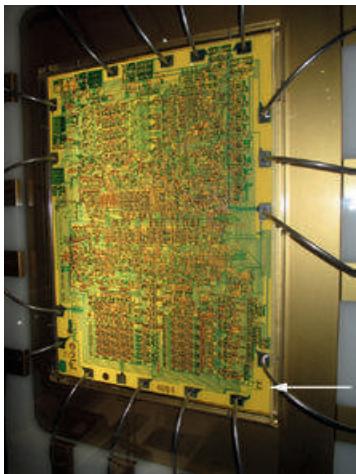


Fig. 7.14. The Intel 4004 microprocessor with Frederico Faggin's initials. This giant model with a 128 \times magnification is on display at the Intel Museum.

In 1965, Moore (B.7.10) wrote an article for the thirty-fifth anniversary issue of *Electronics* magazine entitled “Cramming More Components onto Integrated Circuits,” in which he noted that the complexity of ICs had been doubling every year since 1962. He made the bold prediction that this rate of progress would continue for another decade (Fig. 7.15a), and he speculated that the eventual impact of such chips would be enormous, not only for industry but also for individual consumers: “Integrated circuits will lead to such wonders as home computers – or at least terminals connected to a central computer – automatic controls for automobiles, or portable communications equipment.”¹⁰

Moore made his predictions more than a decade before Steve Jobs and Stephen Wozniak produced the first mass-market personal computer and sixteen years before the appearance of the IBM PC. Caltech engineering professor Carver Mead (B.7.11) dubbed Gordon Moore’s prediction *Moore’s law*, but it took a long time for Moore to get used to using the name! In 1975, Moore updated his law by suggesting that a doubling of the complexity of ICs every two years was more realistic. Nowadays, Moore’s law is usually stated as a doubling of the number of transistors on a chip every eighteen to twenty-four months. This rapid, year-on-year decrease in size of transistors and the corresponding increase in complexity have continued for more than thirty-five years (Fig. 7.15b). Moore reviewed the status of his law again in 1995, at a time when the Intel Pentium microprocessor contained nearly five million transistors. His conclusion was: “The current prediction is that this is not going to stop soon.”¹¹ Today, nearly fifty years after his initial prediction, there are now devices with more than one billion transistors.

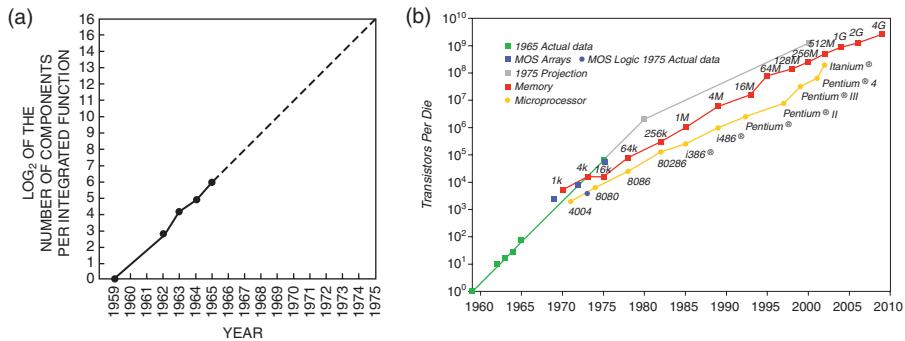
The doubling of complexity embodied in Moore’s law occurs primarily because each generation of semiconductor fabrication facility decreases the minimum feature size on the chip so that the individual transistors can be made smaller. What was not clear in 1965 when Moore made his prediction was whether *quantum tunneling* would prove to be a major limitation on how small the transistors could be made. Quantum tunneling is the process in quantum mechanics in which a quantum particle can tunnel through a barrier in a way that would be impossible for a classical particle. Moore asked Carver Mead at Caltech for advice on this problem. The results of Mead’s investigation were stunning. This is how Mead described the first public presentation of the results of his analysis:



B.7.10. Gordon Moore with Robert Noyce. Moore has a BSc degree in chemistry from UC Berkeley and a PhD from Caltech. He joined Shockley Semiconductor Laboratory in California in 1956 before leaving to found Fairchild Semiconductor Corporation as one of Shockley’s Traitorous Eight. He and co-traitor Noyce later left Fairchild to create Intel in 1968. Moore is probably best known for his observation, originally made in 1965, that the number of transistors on ICs would continue to double every year. Although this increase has slowed to a doubling in eighteen to twenty-four months, Moore’s law has held true for nearly fifty years and is the foundation of the astounding IT revolution we see around us.

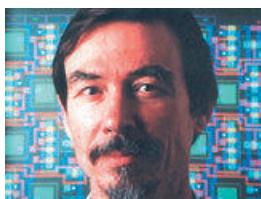
The Computing Universe

Fig. 7.15. Moore's law. (a) Gordon Moore's original prediction. (b) Moore's law still working after nearly fifty years.



In 1968, I was invited to give a talk at a workshop on semiconductor devices at Lake of the Ozarks. In those days you could get everyone who was doing cutting-edge work in one room, so that the workshops were where all the action was. I had been thinking about Gordon Moore's question, and decided to make it the subject of my talk. As I prepared for this event, I began to have serious doubts about my sanity. My calculations were telling me that, contrary to all the current lore in the field, we could **scale** down the technology such that **everything got better**: the circuits got more complex, they ran faster, and they took less power – WOW! That's a violation of Murphy's law that won't quit! But the more I looked at the problem, the more I became convinced that the result was correct, so I went ahead and gave the talk, to hell with Murphy! That talk provoked considerable debate, and at the time most people didn't believe the result. But by the time the next workshop rolled around, a number of other groups had worked through the problem for themselves, and we were pretty much in agreement. The consequences of this result for modern information technology have, of course, been staggering.¹²

The basic scaling principles underlying Moore's law were first described in papers in 1972 by Carver Mead and Bruce Hoeneisen of Caltech and by Robert Dennard and colleagues at IBM. But it was a paper from Dennard in 1974 that laid out the astonishing result – now called *Dennard scaling* – most clearly for the industry. This showed that shrinking the geometry and reducing the supply voltage led to both power reduction and performance improvement. In summary, Dennard scaling said that reducing the length, width, and gate oxide thickness of transistor features by a constant k results in transistors that are k^2



B.7.11. The citation for the 2002 award of the U.S. National Medal of Technology to Caltech professor Carver Mead, reads as follows: "For his pioneering contributions to microelectronics that include spearheading the development of tools and techniques for modern integrated circuit design, laying the foundation for fabless semiconductor companies, catalyzing the electronic design automation field, training generations of engineers that have made the United States the world leader in microelectronics technology, and founding more than twenty companies."¹³



Fig. 7.16. Scanning Electron Microscope image of sub-100nm transistor developed at IBM.

times smaller, k times faster, and dissipate k^2 less power. IBM's MOS memory chips had a feature size of five microns at the time. Dennard and his colleagues projected shrinking the feature size to a fraction of a micron. The use of CMOS technology has allowed IBM to shrink the minimum feature dimension to well below 0.1 micron and enabled IBM to release the Power7 processor in 2010 with 1.2 billion transistors fabricated using a forty-five nanometer process (Fig. 7.16).

As chips grew smaller, not only could more complex chips be designed but also more of them could be produced on a single silicon wafer for the same cost. Moore's law has now held true for nearly fifty years and has been the engine for the vast growth in computing and information processing devices. It was 1970 when Intel produced the first 1,024 bit (1 kilobit) DRAM chip (Fig. 7.17). Only a year later, the first microprocessor, the Intel 4004, was produced with more than two thousand transistors etched in circuits ten microns wide. Just twenty-five years later, in 1995, the industry was producing DRAM chips with sixty-four million bits (64 megabit) and microprocessors like the Pentium with more than four million transistors and a minimum feature size of 0.35 microns. By the turn of the millennium, the industry had moved on to one thousand million bit (1 gigabit) DRAMs and to microprocessors such as the Pentium 4 with more than forty million transistors and a minimum feature size of 0.18 microns. By 2010, the minimum feature size was down to thirty-five nanometers and Intel, AMD, and Nvidia were producing chips with several billion transistors. There will soon be chips that are capable of storing a hundred thousand million bits – more bits than there are stars in our galaxy!

Powerful computers are now needed to design each new generation of chips: literally, we are using our present-day computers to design the next generation of computers. The international silicon industry produces the “Semiconductor Roadmap” that examines the engineering and design challenges required to keep on the track of Moore's law. Although the charts of the Semiconductor Roadmap boldly carry forward Moore's law many years into the future, there are many significant technical problems to be solved along the way. We will look at some possible solutions in Chapter 15.

Finally, in addition to these technical challenges, there is an economic one: the sheer cost of building the manufacturing facility for each new generation of chips (Fig. 7.18). Arthur Rock, one of the investors who helped Moore and Noyce raise funding to start Intel, is credited with Rock's law, which states: “A very small addendum to Moore's Law which says that the cost of capital equipment to build semiconductors will double every four years.”¹³ It was this spiraling cost of fabrication facilities that led Morris Chang, a Taiwanese engineer and entrepreneur, to pioneer the concept of a silicon foundry – essentially a “fab-for-hire.” Companies can do their own chip design and then pay the foundry to manufacture their chips. Chang set up the Taiwan Semiconductor Manufacturing Company (TSMC) in 1987. It is now the world's largest foundry with more than \$13 billion in revenue. According to James Plummer, Dean of the School of Engineering at Stanford University:

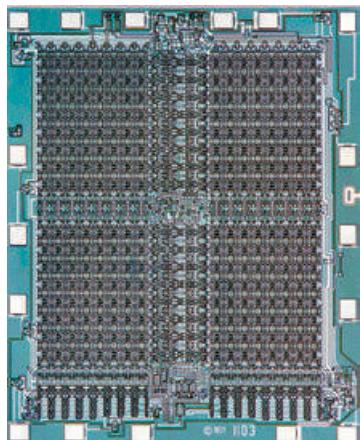
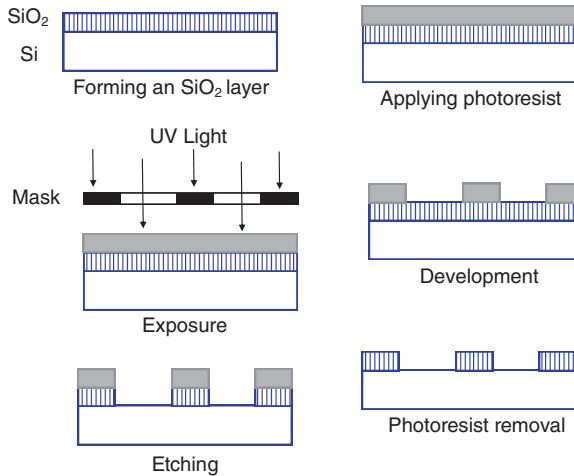


Fig. 7.17. Intel's revolutionary 1103 memory chip. One of the first customers was Xerox PARC where Chuck Thacker and Butler Lampson used the temperamental chip for the memory of famous Alto personal computer (see next chapter).

Fig. 7.18. ICs are produced by a complicated process called photolithography. The technique is similar to traditional photography but in this case we use a silicon wafer instead of a film of emulsion.



Morris Chang completely changed the landscape of the semiconductor industry. He enabled start-ups to start with a few million dollars rather than a few hundred million. That makes a huge difference.¹⁴

Jen-Hsun Huang, a co-founder of Nvidia, credits TSMC with enabling all sorts of creative ideas in areas such as networking, consumer electronics, computers, and automotive technology to be turned into successful companies because “the barriers to getting your chips built, to realizing your imagination, disappeared.”¹⁵

The end of the free lunch: parallel computing and the multicore challenge

As Moore’s law predicted, designers and manufacturers have delivered smaller, faster chips requiring less power for more than four decades. But we have now reached the length scale at which the transistor’s gate insulator is only a few atoms thick. Because the transistor is not a perfect switch, it leaks some current even when it is in the turned off state. As the transistor size decreases, if we continue to scale down the voltage, the current leakage increases exponentially. To keep this leakage under control, the voltage can no longer be scaled down with the dimensions of the chip. We can still shrink the size of the transistors and place more of them on a chip, but they will not be much faster than current generation transistors because the insulating silicon dioxide layer cannot get thinner and the power consumption of the chips limits our ability to clock the chip as fast as we could. As a result, chip architects have developed *multicore* architectures with multiple CPUs integrated on a single chip. Dual-core chips have been in widespread use for some time now, and quad-core chips are increasingly common. Eight-core chips are now available and the industry is experimenting with chips containing tens or even hundreds of cores (Fig. 7.19).

Performance improvement must now come from writing software that uses multiple cores together to solve a problem. For many types of applications it is

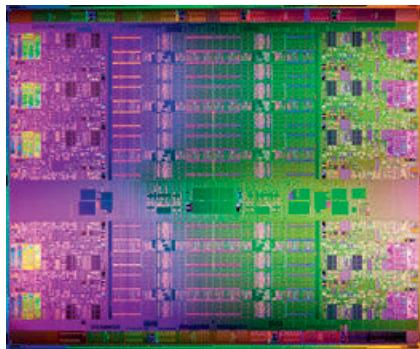


Fig. 7.19. The Intel Xeon E7 processor contains ten cores and a total of 2.6 billion transistors.

easy to make use of the parallelism of multiple cores. Such problems are said to exhibit *embarrassingly obvious* parallelism and, although obvious, this type of parallel computing application is very common. It is much more difficult to get speed-up by parallelizing a single application and then distributing the required computation over the multiple cores. Figure 7.20 illustrates three common types of parallelism.

One challenging problem that chip designers now face is the problem of *dark silicon*. This is the fact that although we will be able to make devices with many more transistors than today's chips, we will not be able to power all of them simultaneously due to power density limitations on the chip. Engineers are now actively looking for new ways to reduce power consumption in their chip designs. Multicore chips – requiring parallel programming – are therefore only a short-term solution to the problem of providing more performance per chip. In April 2005, Moore stated in an interview that it was clear that his law cannot be sustained indefinitely:

In terms of size [of transistors] you can see that we're approaching the size of atoms which is a fundamental barrier, but it'll be two or three generations before we get that far – but that's as far out as we've ever been able to see. We have another 10 to 20 years before we reach a fundamental limit. By then they'll be able to make bigger chips and have transistor budgets in the billions.¹⁶

Unless we can come up with some radically new processor technologies, the end of Moore's law is in sight! In Chapter 15 we look at some possible solutions.

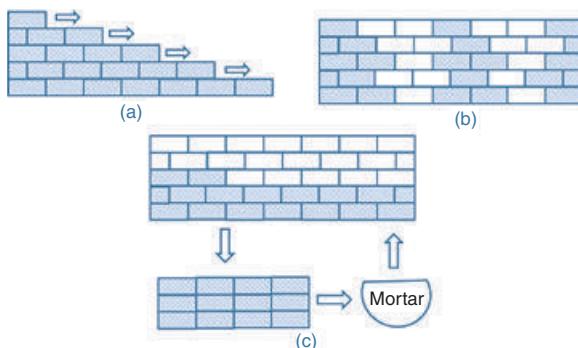


Fig. 7.20. Parallel Computing Paradigms illustrated by Fox's wall construction analogy: (a) "pipeline" parallelism, (b) "domain" parallelism, (c) "task" parallelism. (a) Pipeline parallelism in which each bricklayer is responsible for laying one row of bricks. Obviously the bricklayer for the second row cannot start until the first bricklayer has laid some bricks. Similarly, when the first bricklayer has finished the top row of bricks, the others are still finishing. This is a good analogy for the parallelism used in vector supercomputers (see section on supercomputers at the end of this chapter). (b) In domain parallelism, each bricklayer is responsible for a given section of the wall. Obviously at the edges of these domains the two bricklayers need to coordinate their activity. This is a good analogy for the parallelism used in distributed memory microprocessor-based parallel supercomputers. (c) Task parallelism is where each bricklayer is free to collect a brick and put it anywhere in the wall. It is a good analogy for the very common "embarrassingly obvious" parallelism of many types of application.

Key concepts

- Metals, insulators, and semiconductors
 - Band theory
 - Doped semiconductors: p-type and n-type
- Transistors
 - Point-contact and junction transistors
 - Mesa and planar transistors
- Integrated circuits
 - MOSFET transistors and CMOS technology
 - Random access memory: DRAM, SRAM, and ROM
- Moore's law
 - Microprocessors
 - Dennard scaling
 - Fabrication facilities and silicon foundries
- Multicore chips
 - Parallel computing
 - Dark silicon



A cartoon from Gordon Moore's original 1965 paper on Moore's law. Remarkably, Moore envisaged home computers being sold as a commodity long before the microprocessor gave birth to the personal computer (see next chapter).

A quantum theory primer

In the first half of the twentieth century, our understanding of matter underwent a profound revolution with the advent of quantum theory. Although a deep understanding is not needed for a reader's comprehension of this book, this section summarizes the essence of quantum theory that now underpins all of modern physics. This primer is not intended as a substitute for learning more about quantum theory but will be helpful in our understanding of the semiconductor materials that are central to the modern computer industry and of attempts to develop a new type of "quantum computer."

Although it is only about one hundred years old, quantum theory helped settle a scientific debate about the nature of light that began in the seventeenth century. Was light best described as a stream of particles, as Isaac Newton claimed, or was light some form of wave motion, as the Dutch physicist Christiaan Huygens had proposed? In 1801, the English physicist Thomas Young demonstrated that when two rays of light meet, they form a series of bright and dark bands called an *interference pattern*. Since these patterns are characteristic of waves – like ripples on a pond – the nature of light seemed to be settled. Then in 1921, Albert Einstein won the Nobel Prize for his explanation of the "photoelectric effect," the emission of electrons by a metal when exposed to light. Einstein found that light is made up of particle-like packets of energy called photons.

The theory of quantum mechanics emerged in the 1920s, pioneered by physicists such as Werner Heisenberg, Erwin Schrödinger, and Paul Dirac. Quantum mechanics provided successful "explanations" – in terms of its predictions – of the behavior of light, electrons, atoms, and nuclei in the microscopic world of atoms and nuclei (see Appendix 1). But there is a price to pay for this success: objects like photons and electrons behave in an essentially quantum mechanical way. All we can know about their motion is described by the evolution of a "probability wave." The wave equation discovered by Schrödinger describes how the probability wave for a quantum object – usually represented by the Greek letter ψ – evolves with time. We can only observe probabilities and, according to quantum theory, it is the square of this wave amplitude ψ that gives us the probability that we will observe the object at any given place and time.

Despite this emphasis on probability and uncertainty – epitomized by Heisenberg's famous "uncertainty principle" – quantum mechanics is the only theory capable of making accurate predictions for systems of atomic sizes or smaller. In addition, it is the very certainties of quantum mechanics that are responsible for the existence of the different chemical elements we see around us! According to quantum theory, electrons bound to an atom can only have certain energies. We can see how this comes about as follows. The problem of finding the allowed energies of an electron in an atom is analogous to finding the allowed energy levels of a charged particle in a potential well. In real life, we have to solve the Schrödinger wave equation in three-dimensional space but we can get some idea of the quantum solution for an atom by considering the problem of finding the allowed energy levels of an electron confined to a one-dimensional box. In the classical world, the electron in a box could have any energy; in the quantum world, the wave patterns must match the dimensions of the well – like finding the allowed wavelengths for a violin string. This means that only certain energies are allowed for the electron in a box (Fig. 7.21).

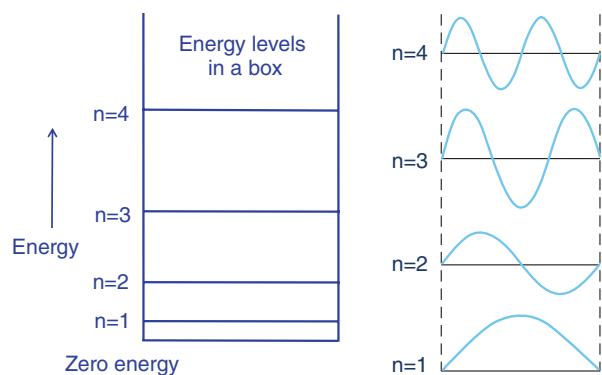


Fig. 7.21. Energy levels and wave functions for electrons confined to a box. (a) Energy levels for a quantum particle in a box. The energy levels are labeled by the quantum number "n." (b) This shows how the corresponding wave forms fit into the box. The wave function has to be zero at the edges.

The other two ingredients that we need from quantum theory to gain an understanding of the Periodic Table of the elements are *electron spin* and the *Pauli Exclusion Principle*. Electrons have “spin,” somewhat like a spinning top, but unlike a classical top, an electron can only exist in one of two spin states, called “up” and “down.” Pauli’s Exclusion Principle says that only one electron is allowed in each quantum state. This means that in our electron potential box (Fig. 7.22) we can only put two electrons in the lowest energy state, called the *ground state*: one electron with spin up and the other with spin down. If we want to add another electron to the box, we have to give it more energy and place it in the next energy level – called the *first excited state*.

It is the exclusion principle – insisting that electrons have to occupy distinct quantum states – that gives the stability and volume of ordinary matter. As Richard Feynman says: “It is the fact that electrons cannot all get on top of each other that makes tables and everything else solid.”¹⁷ The exclusion principle applies to all “matter-like” quantum objects such as electrons, protons, and neutrons. For “radiation-like” objects such as photons, the exclusion principle does not apply, and we can put as many photons as we like into the same quantum state. This has led to amazing applications such as lasers and superconductivity.

Armed with these fundamental quantum concepts, we are now able to explain the difference between metals, semiconductors, and insulators. In a later chapter we will see how these quantum ideas are being used to build a new type of quantum computer.

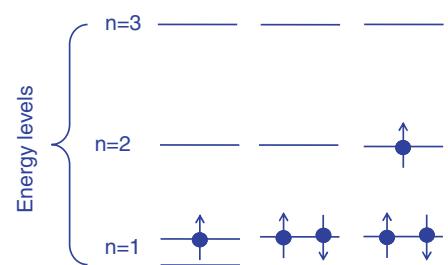


Fig. 7.22. Electrons in a box. The electrons can only fill up the energy levels according to Pauli’s Exclusion Principle, which states that only one electron is allowed to occupy a quantum state. Each quantum level can therefore accommodate two electrons – one with spin up and one with spin down. The n = 1 level can therefore only accommodate two electrons: the next electron must go into the more energetic, first “excited” state, n = 2.

Supercomputers

Cray computers were the first commercially successful supercomputers. They were designed and constructed in the small remote town of Chippewa Falls in Wisconsin. As we have seen in Figure 2.20, fetching data from main memory takes many more clock cycles than the number of cycles required to perform an operation on data that is already in the registers. For programs that involve operations on vectors – a one-dimensional array of numbers – Seymour Cray (B.7.12) realized that it would be possible to set up a pipeline that can hide much of the latency caused by fetching the data from the main memory and getting a new instruction for each operation. For example, if we need to multiply two vectors, we need to apply the same operation on each pair of elements of the vectors. We can therefore arrange things so that while the CPU is multiplying the first two elements of the two vectors, the computer is already fetching the next two elements to be multiplied. This is an implementation of pipeline parallelism as illustrated in Figure 7.20a. Roger Hockney, a British computer scientist who pioneered serious benchmarking of parallel computers, characterized vector supercomputers in terms of a parameter he called “ $n_{1/2}$ ” – this is the length of the vector that was required for the supercomputer to reach half of its maximum speed. It is essentially a measure of the distance of main memory from the registers in terms of cycles. The success of the Cray-1 was due to its much smaller $n_{1/2}$ than its rival supercomputers. This meant that it was able to achieve high speeds on problems requiring much shorter vector operations.

Today, the parameters of Cray-1 look almost laughably slow but in the 1970s the Cray-1 was at the cutting edge, with a clock speed of 80 megaHertz (MHz) and an 8 megabyte (Mbyte) main memory (Fig. 7.23). To minimize signal delays, the frame of the computer was bent into a C shape, thus enabling the use of shorter wires. The speed was eighty million operations per second on floating point numbers or eighty megaflop/s (Mflop/s). The cooling and power distribution required many ingenious solutions. The Cray-1 used liquid Freon instead of water for cooling, and copper plates between circuit boards. No wonder Cray joked that becoming a plumber was the peak of his career:

I made square cabinets with glass doors and aluminum and imitation walnut trim [CDC 7600] and cylindrical cabinets [CDC 8600 and CRAY-1]. After a while, I got tired making cabinets and so I decided I needed to go into a new profession. Now, I am into plumbing [CRAY-2, 3]. It is a lot less prestige, but I am making even more money.¹⁸

Cray supercomputers were expensive and only automotive firms and large national laboratories could afford them. The first Cray-1 was installed in Los Alamos in 1976. The applications were weapon simulations, weather prediction, and cryptoanalysis. Each new Cray machine was shipped with a case of Leinenkugel's beer, also a product of Chippewa Falls. In the automotive industry, they were used for the first car-crash simulations.



B.7.12. The name of Seymour Cray (1925–96) has become inseparable from supercomputers. Cray was one of the first to recognize that maximizing the speed of moving data between the processor and memory and hiding memory latency using a vector pipeline was the key to achieving high-performance computing speeds. He is pictured here next to a Cray-1 supercomputer.



Fig. 7.23. A photograph of two Cray-1 supercomputers at Lawrence Livermore National Laboratory. The upholstered bench around the computing tower hides the power supply and the intricate network of Freon cooling pipes. The supercomputer was sometimes referred to as “the most expensive seat in the world.”

Such simulations have now become an essential element of car design with detailed crash simulation using models of the driver and passengers.

By the 1980s microprocessors based on new transistor technology (CMOS) started to appear as alternative building blocks of supercomputers. Cray was rather skeptical and when he was asked whether he had considered building the next generation of Cray computers on the new components he famously said “If you have a heavy load to move. What would you rather use a pair of oxen or hundred of chicken.”¹⁹

However, in the early 1980s, Geoffrey Fox and Chuck Seitz at Caltech put together a parallel computer called the Cosmic Cube. In essence, this was a collection of IBM PC boards, each with an Intel microprocessor and memory, connected together by a so-called *hypercube network*. The importance of the Cosmic Cube experience was that Fox and Seitz demonstrated for the first time that it was both feasible and realistic to use such “distributed memory” parallel computers to solve challenging scientific problems. In this case, programmers need to exploit domain parallelism as shown in Figure 7.20b. Instead of the latency caused by filling and emptying a vector pipeline, the overhead in such distributed memory programs comes from the need to exchange information at the boundaries of the domains since the data is subdivided among the different nodes of the machine. This style of parallel programming is called *message passing*.

Today, all the highest performance computers use such a distributed memory, message passing architecture, albeit with a variety of different types of networks connecting the processing nodes. Instead of the Cray-1’s eighty Mflop/s peak performance, we now have distributed memory supercomputers with top speeds of teraflop/s and petaflop/s, while gigaflop/s performance is now routinely available on a laptop! The supercomputing frontier is to break the exaflop/s barrier and there are now U.S., Japanese, European, and Chinese companies taking on this challenge. In answer to Cray’s sarcastic comment about hundreds of chickens, Eugene Brooks III, one of the original Cosmic Cube team at Caltech, characterized the success of commodity-chip based, distributed memory machines as “the attack of the killer micros.”²⁰

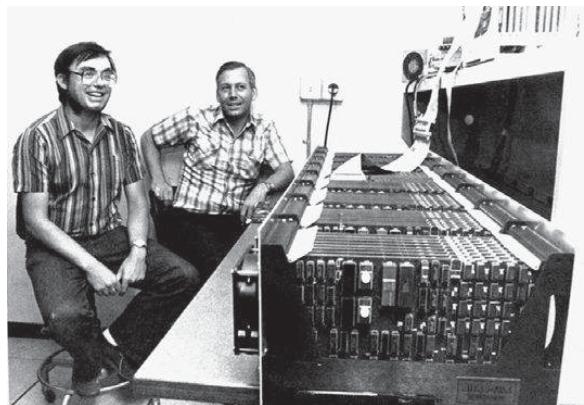


Fig. 7.24. Geoffrey Fox and Chuck Seitz pictured next to the Caltech Cosmic Cube machine. This parallel computer became operational in October 1983 and contained sixty-four nodes each with 128 kilobyte memory. A computing node was made up from an Intel 8086 processor with an 8087 coprocessor for fast floating-point operations. The nodes were linked together in a so-called hypercube topology – several cubes connected together – for minimizing communication delays between nodes. The size of the computer was only six cubic feet and drew less than a kilowatt of power. The Cosmic Cube and its successors represented a serious challenge for the Cray vector supercomputers. Today all the highest-performing machines use a distributed memory message-passing architecture similar to the Caltech design.

8 Computing gets personal

I think it's fair to say that personal computers have become the most empowering tool we've ever created. They're tools of communication, they're tools of creativity, and they can be shaped by their user.

Bill Gates¹

The beginnings of interactive computing

In the early days of computing, computers were expensive and scarce. They were built for solving serious computational problems - and certainly not for frivolous activities like playing games! The microprocessor and Moore's law have changed this perspective – computing hardware is now incredibly cheap and it is the software production by humans and management of computers that is expensive. Some of the ideas of interactive and personal computing can be traced back to an MIT professor called J. C. R. Licklider. Lick – as he was universally known – was a psychologist and one of the first researchers to take an interest in the problem of human-computer interactions. During the Cold War in the 1950s, he had worked at MIT's Lincoln Labs on the Semi-Automated Ground Environment (SAGE) system designed to give early warning of an airborne attack on the United States. This system used computers to continuously keep track of aircraft using radar data. It was this experience of interactive computing that convinced Lick of the need to use computers to analyze data as the data arrived – for “real time” computing.

Another type of interactive computing was being developed at around the same time. Engineers at MIT's Lincoln Labs had developed the TX-0 in 1956 – one of the first transistorized computers. Wesley Clark and Ken Olsen had specifically designed and built the TX-0 to be interactive and exciting, the exact opposite of sedate batch processing on a big mainframe computer. Olsen recalled:

Then we had a light pen, which was what we used in the air-defense system and which was the equivalent of the mouse or joystick we use today. With that you could draw, play games, be creative – it was very close to being the modern personal computer.²

This level of interactivity, and for what then seemed to be frivolous uses of valuable computing time, was a far cry from the regimented bureaucracy of batch processing. To popularize their ideas, Olsen and Clark decided to send

Fig. 8.1. Eager to be known as more than a supplier of office copiers, Xerox created PARC in 1970. PARC's mission was to create the "Office of the Future." George Pake and Bob Taylor assembled a team of world-class scientists and engineers – to create the "architecture of information" – and PARC became a hothouse of innovation that flourished for decades. The atmosphere at Xerox PARC reflected the laid-back, West Coast, hippie-influenced culture of the 1970s. It was worlds apart from the culture of Xerox's corporate headquarters in Connecticut. In an unrivaled burst of creativity, the PARC researchers developed most of the personal computing environment that is still with us today – overlapping windows, GUIs, Ethernet, digital video, word processing, and laser printers. Although PARC's inventions never led to a successful personal computer business for Xerox and many ideas never became successful commercial products, the laser printer alone generated billions of dollars in sales for Xerox, much more than their total investment in PARC.



the TX-0 from their off-campus Lincoln Laboratory site over to the main MIT campus. Clark later wrote:

The only surviving computing system paradigm seen by MIT students and faculty was that of a very large International Business Machine in a tightly sealed Computation Center: the computer not as a *tool*, but as a *demigod*.

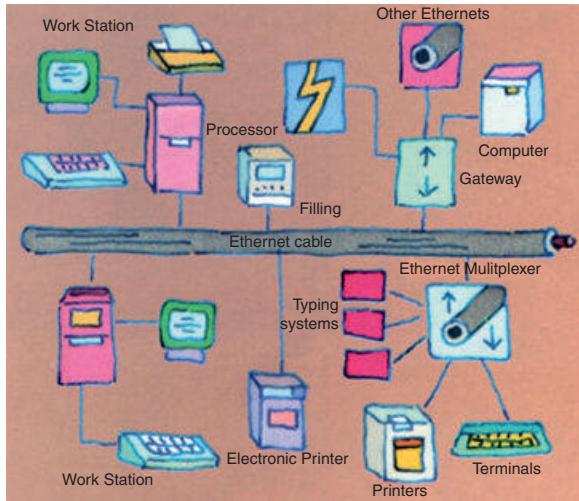
Although we were not happy about giving up the TX-0, it was clear that making this small part of Lincoln's advanced technology available to a larger MIT community would be an important corrective step.³

Yet a third type of interactive computing was also being experimented with at MIT. As we have seen, John McCarthy had become so frustrated with this remote, batch processing model of computing that he had come up with the idea of time sharing. Sharing the computing cycles of a single large computer among several users, each connected to the computer with their own terminal, introduced a different type of interactivity – one in which the user had the illusion of being the sole user of the computer.

It was out of this hotbed of experimentation with interactive computing at MIT that Lick was recruited in 1962 to lead a new computer research program at the U.S. Department of Defense's Advanced Research Projects Agency (ARPA). When he arrived at the Pentagon, Lick set about creating a major research program in interactive computing – and in so doing laid the foundations for much of the university computer science research in the United States. As we will see in [Chapter 10](#), Lick also had the idea of connecting remote computers together to create what later became the ARPANET – although it was left to Bob Taylor, one of Lick's successors at ARPA, to get the funding to implement Lick's vision.

How did we get from these early explorations of interactive computing to the personal computing we see around us today? In his book *Dealers of Lightning: Xerox PARC and the Dawn of the Computer Age*, Michael Hiltzik highlights the contribution of the researchers at Xerox Corporation's Palo Alto Research Center (PARC) ([Fig. 8.1](#)):

Fig. 8.2. A concept sketch for Metcalf and Bogg's Ethernet. Their original Ethernet report observed: "Just as computer networks have grown across continents and oceans to interconnect major computing facilities ... they are now growing down corridors and between buildings to interconnect minicomputers in offices and laboratories."¹



Every time you click a mouse on an icon or open overlapping windows on your computer screen today, you are using technology invented at PARC. Compose a document by word processor, and your words reach the display via software invented at PARC. Make the print larger or smaller, replace ordinary typewriter letters with a Braggadocio or Gothic typeface – that's technology invented at PARC, as is the means by which a keystroke speeds the finished document by cable or infrared link to a laser printer. The laser printer, too, was invented at PARC.⁴

Why then was Xerox not at the heart of the personal computer revolution? The answer is complicated, but ultimately Xerox failed to fully exploit the amazing inventions of its PARC researchers and missed a huge opportunity to create a new computing paradigm. Nevertheless, just one of PARC's inventions, the laser printer, generated billions of dollars in revenue for the company, many times more than its total investment in PARC (Fig. 8.2). But there could have been so much more (Fig. 8.3). This wonderful burst of creativity at Xerox PARC took place in the early 1970s. The personal computer revolution arrived by a different route and was triggered by the arrival of cheap and powerful microprocessors, an enthusiastic community of computer hobbyists, and four remarkable young entrepreneurs without a university degree between them!

The Altair and Microsoft

In January 1975, an editorial in the magazine *Popular Electronics* proudly announced the arrival of the world's first "home computer" (Fig. 8.4):

For many years, we've been reading and hearing about how computers will one day be a household item. Therefore, we're especially proud to present in this issue the first commercial type of minicomputer project ever published

Fig. 8.3. Taylor and PARC scenes. Clockwise from top left: Bob Taylor, Alan Kay, the Dynabook, the pocket calculator, and *Rolling Stone* reporter Stewart Brand drawing with a computer.

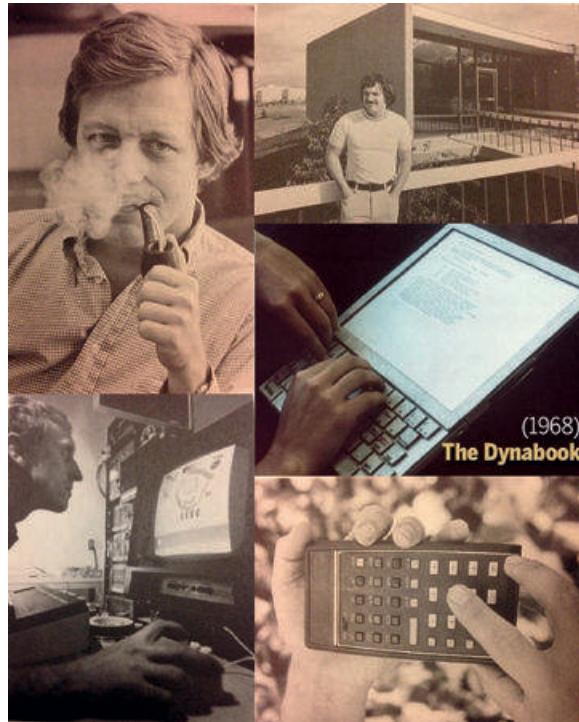


Fig. 8.4. The launch of the Altair by *Popular Electronics* in January 1975. This cover featuring the MITS Altair 8800 excited thousands of hobbyists eager to build their own computers. Roberts had not finished the design of the computer when editor Les Solomon solicited the article. The magazine chose the name Altair after the star. The prototype machine sent by Ed Roberts got lost in the mail and never arrived so the magazine photographed an empty box for the cover photo.

that's priced within reach of many households – the Altair 8800, with an under \$400 complete kit cost, including cabinet.⁵

On the cover was a picture of the Altair 8800 computer, complete with flashing lights. In actual fact, it was just a picture of an empty case: the first prototype had gone missing in a shipment between Albuquerque and New York, and there had not been time to assemble another machine and get it to New York before the magazine's deadline. The Altair was the brainchild of Ed Roberts, a U.S. Air Force electronics engineer, and his small electronics hobby-kit company called Micro Instrumentation and Telemetry Systems (MITS), based in Albuquerque, New Mexico (Fig. 8.5). His company had been one of the first to put a calculator kit on the market, but by 1974 fully assembled calculators were selling for less than the kit. To save the company, Roberts devised a totally new product – a computer kit based around the latest 8080 microprocessor from Intel. This chip was more powerful than its predecessor, the 8008, and, as historian Paul E. Ceruzzi says, “[it] required only six instead of twenty supporting chips to make a functional system.”⁶ In his design Roberts introduced an “open bus architecture” to allow for the addition of extra circuit boards. A *bus* is just a set of connections linking all the major components of the machine, including the central processor, memory, and input/output (I/O) devices, in a standard way. The *bus architecture* makes it possible to customize the computer with circuit boards offering a specific functionality. If a user wants a better sound system, for example, he or she is able simply to unplug the old sound card from the bus and plug in a new one.

MITS planned to produce and sell plug-in cards for *peripheral devices*, auxiliary devices that would connect to the computer, such as memory boards, paper-tape readers, terminals, and printers. Making the bus design open was



Fig. 8.5. The MITS Altair 8800. Most do-it-yourself hobbyists did not want to have to buy all the integrated circuits and other components needed to build the computer. The appeal of the Altair 8800 was that it was the first build-your-own-computer kit.

important because it would allow hobbyists and other electronics companies to make cards for the Altair. In spite of the fact that no peripheral cards were available for many months after launch, and there was no prepackaged software available for the machine, customers deluged MITS with orders for the Altair. But the machine was far from being a household item: to get the Altair to actually do anything, a user had to painstakingly enter a program by hand, bit by bit, using toggle switches on the front panel. It was clear from the time of its launch that what the Altair urgently needed was the capability to run a high-level programming language.

The BASIC programming language was developed in the 1960s by John Kemeny and Thomas Kurtz at Dartmouth College. BASIC is an acronym for Beginner's All-purpose Symbolic Instruction Code, and the language was used at Dartmouth for teaching undergraduates. A team of engineers at Digital Equipment Corporation (DEC) had taken an important step in the evolution of BASIC in 1971. They used BASIC to implement a new operating system for the PDP-11 minicomputer, and they extended and modified the language in a number of important ways. The most important was the introduction of "PEEK" and "POKE" commands, which gave programmers the ability to execute low-level system calls, and to interact with the computer's memory at the byte level directly from a BASIC program. The engineers also made some compromises to the original rules of the language, changes disapproved of by Kemeny and Kurtz, that allowed DEC BASIC to be used on machines with very limited memory. Despite its ease of use, academic computer science departments often discouraged BASIC as a teaching language because it was believed to encourage bad programming habits. Edsger Djikstra, who received the Turing Award for his contributions to computing, went so far as to say that programming in BASIC causes brain damage (see B.4.11). For the personal computing revolution, BASIC, with the DEC extensions allowing programmers to pass easily from BASIC to machine code, was the obvious first choice. Ed Roberts said that he had settled on BASIC for the Altair because you "could teach any idiot how to use [it] in no time at all."⁷⁷

Paul Allen and his friend from high school, Bill Gates, had been entranced by computers from their days at Lakeside School, a private school in Seattle (B.8.1). In their spare time, they had worked as testers for the C-Cubed computer



B.8.1. The photograph shows Paul Allen at a teletype and Bill Gates (standing) when they were at Lakeside School in Seattle in 1968. Allen and Gates had signed up for the school's independent study option on programming and learned to program in BASIC. The two became captivated by computing and spent many hours of their spare time working for a local computer company. As a result of this experience, they became proficient in operating system software and assembly language for the PDP-10. The appearance of the Altair do-it-yourself computer kit on the cover of the magazine *Popular Electronics* in January 1975 excited Allen and Gates. They contacted Ed Roberts, the designer of the Altair, and offered to produce a BASIC interpreter for the machine. Remarkably, they had no access to an Altair machine when they wrote their interpreter. Instead, they debugged their BASIC interpreter using a simulator of the Intel 8080 microprocessor that Allen wrote for the PDP-10 that Gates had access to at Harvard. Aided by fellow Harvard student Monte Davidoff, the three of them finished their BASIC interpreter in just eight weeks. When Harvard reviewed the usage statistics of their PDP-10 machine during January, they found that William Henry Gates III had used a surprisingly large amount of computer time!



Fig. 8.6. The interpreter source tape for Altair BASIC. Paul Allen finished the software while flying to Albuquerque to demonstrate the interpreter to Ed Roberts and his engineers at MITS. Microsoft later created interpreters for many other languages and processors, although BASIC remained its most valuable product into the early 1980s. The text on the tape reads “BASIC 8K without cassette July 2 1975.”



B.8.2. Ed Roberts (1941–2010) founded the Micro Instrumentation and Telemetry Systems (MITS) company in 1970, and initially produced electronics kits for model rockets, and later, for calculators. When calculators became too cheap for the MITS kits to be profitable, Roberts designed a \$397 “personal computer,” do-it-yourself kit called the Altair 8800. After the January 1975 issue of *Popular Electronics*, orders began to pour in and the Altair became the catalyst for the personal computer revolution.

company in exchange for free use of the company’s new PDP-10 minicomputer. There they learned new programming skills from the company’s expert programmers. One of them, Steve Russell, had worked with John McCarthy at MIT and had developed Spacewar!, one of the first interactive computer games. In those early days, computer companies placed most value on their hardware: software came free as an inducement for customers to buy the machine. As a result, C-Cubed had access to the source code of the TOPS-10 operating system, developed by DEC for the PDP-10 mainframe computer, and C-Cubed was working to debug and enhance the system. Russell noticed Allen’s interest in learning more about programming and introduced him to the PDP-10 assembly language. As a project, he suggested that Allen try to improve and enhance the BASIC compiler for the PDP-10. When C-Cubed closed down, Allen and Gates continued their projects by relocating unofficially to the computer science lab at the University of Washington. During the next few years, they also worked as programmers on various commercial contracts, writing code for PDP-10 machines. In the summer of 1972, they formed a partnership called Traf-O-Data to develop both the hardware and software to automate the measurement of traffic flows using Intel’s newest microprocessor, the 8008, to do the data analysis. They persuaded Paul Gilbert, an engineering student at the University of Washington, to design and build the hardware. To write the software, because the hardware did not yet exist, they decided to simulate the 8008’s instruction set on a PDP-10 minicomputer. Although Traf-O-Data was not a commercial success, Allen and Gates built an unrivaled set of development tools for the 8008 microprocessor. These tools included an assembler, to translate from assembly language into machine code; a simulator, to model and study real-life situations on the computer; and a debugger that allowed the programmer to stop the program in mid-execution.

By December 1974, Bill Gates had gone to Harvard, and Paul Allen had also moved to the Boston area working as a programmer for Honeywell. When Allen came across the January issue of *Popular Electronics*, the two friends realized that their experience had uniquely prepared them for the challenge of writing BASIC for the Altair. Allen describes Gates calling Ed Roberts (B.8.2) in Albuquerque, pretending to be Paul Allen:

“This is Paul Allen in Boston,” Bill said. “We’ve got a BASIC for the Altair that’s just about finished, and we’d like to come out and show it to you.” I admired Bill’s bravado but worried that he’d gone too far, since we’d yet to write the first line of code.⁸

Roberts had received many calls from people making similar claims. He told Gates that he would give a contract to the first person to demonstrate a BASIC that actually worked on the Altair.

With this as encouragement, Allen and Gates bought an 8080 instruction manual and set about extending their Traf-O-Data development tools for the new microprocessor. Gates led the design of the BASIC interpreter (Fig. 8.6). A compiler converts the entire source code of a program into an assembly language program in one operation: an interpreter translates and executes small pieces of source code at a time and therefore takes up much less memory. To

write the code for the decimal arithmetic operations required for BASIC, they recruited a fellow Harvard student named Monte Davidoff. That January and February, the three worked until late every night and through all the weekends. Their BASIC interpreter was finished in just eight weeks, and Allen flew to Albuquerque for its first encounter with the real Altair hardware. To the amazement of Roberts and his engineers, the 8080 BASIC interpreter developed by Gates and Allen ran the first time. The two friends signed a licensing agreement with MITS in July 1975 and needed a name for their partnership. They decided on Micro-Soft, for *Microprocessors and Software*, although they were not consistent about having the hyphen. In November 1976, the name of their company was registered with the state of New Mexico as Microsoft Corporation (B.8.3).

It was the unique technical experience of Allen and Gates, together with their PDP-10 simulator and development tools, that enabled them to beat seasoned professional software engineers and university computer scientists in developing the first usable software for the Altair. Their BASIC interpreter packed many features and impressive performance into a very small amount of memory. Paul Ceruzzi summarized their achievements:

The BASIC they wrote for the Altair, with its skillful combination of features taken from Dartmouth and from Digital Equipment Corporation, was the key to Gates's and Allen's success in establishing a personal computer software industry.⁹

By 1979, Microsoft's BASIC interpreter became the first microprocessor software product to surpass a million dollars in sales (Fig. 8.7).

The Homebrew Computer Club and Apple

The arrival of the Altair stimulated the electronic hobbyist community to make microprocessor-based personal computers a reality. Computer clubs sprang up all over the United States including, most famously, the Homebrew Computer Club in Silicon Valley. In the early years of the personal computer,



B.8.3. This photograph of thirteen of the original fifteen Microsoft staff was taken in Albuquerque on 7 December 1978. Top row, left to right: Steve Wood, programmer; Bob Wallace, production manager-designer; Jim Lane, project manager. Middle row, left to right: Bob O'Rear, mathematician; Bob Greenberg, programmer; Marc McDonald, programmer and Microsoft's first employee; Gordon Letwin, programmer. Bottom row, left to right: Bill Gates, cofounder; Andrea Lewis, technical writer; Marla Wood, bookkeeper, married to Steve Wood; and Paul Allen, cofounder. Allen left Microsoft in 1983 and is now owner of the Seattle Seahawks, winners of the 2014 NFL Super Bowl. Two employees were not in the photograph. Ric Weiland was house hunting in preparation for Microsoft's move to Seattle, and Miriam Lubow was unable to make it into town for the photograph because of a rare snowstorm in Albuquerque that day.

Fig. 8.7. An aerial view of the present Microsoft campus in Redmond, near Seattle in Washington State.



from 1975 to 1978, hobbyists played a crucial role in its development, while the chip manufacturers and traditional computer companies focused on the business computer market. Chip suppliers were developing a market for microprocessors designed to handle control functions within larger systems – the *embedded systems* market. IBM, DEC, and other computer companies were focused on mainframes or mini-computers and had not embraced the idea of a truly personal computer. Only enthusiastic hobbyists were willing to put up with the difficulties of programming such primitive microprocessor systems like the Altair at a time when there were no peripheral devices available to make the system easier to use. Fortunately, the open bus architecture of the Altair meant that electronic hobbyists as well as other companies besides MITS were soon able to create these components and have a stake in this nascent industry.

Although IBM had started “unbundling” its hardware and software – that is, selling its hardware and software separately – as early as 1968, the original tradition of hardware manufacturers was for them to give the software away for free as an added feature of their machines. This practice led to a schism in the computing community that to some extent persists to this day. Allen and Gates were surprised and disappointed when they found that their royalty check for Altair BASIC in 1975 was only \$16,005. Less than one in ten Altair owners was actually purchasing their BASIC software, instead relying on a tradition of widespread copying. This led to the famous “Open Letter to Hobbyists” from Bill Gates, published in the newsletter of the Homebrew Computer Club, in which Gates argued that unauthorized copying discouraged the development of high-quality software. The article generated a heated debate in the hobbyist community (Fig. 8.8).

The arrival of the Altair inspired the founding of the Homebrew Computer Club. The first meeting took place in March 1975 in a garage in Menlo Park, California, and subsequent monthly meetings were held in the auditorium of the Stanford Linear Accelerator Center. Among the thirty or so attendees at the first meeting – the numbers later grew to several hundred – was Stephen Wozniak, or Woz as he was known to his friends. Although he had dropped out of formal university education, Woz was an exceptionally talented computer engineer who worked in the calculator division of the Hewlett-Packard



Fig. 8.8. A meeting of the Homebrew Computer Club. The club met in the auditorium of the Stanford Linear Accelerator Center and hobbyists were encouraged to display their latest creations in the entry lobby. Anyone who attended even once was considered a “member” and could sign up for the newsletter. Founding member Fred Moore published the first issue of *Homebrew Computer Club Newsletter* on 15 March 1975. Moore expressed the shared excitement of the group: “I expect home computers will be used in unconventional ways – most of which no one has thought of yet.”^{f2}



Fig. 8.9. Steve Wozniak demonstrated the prototype Apple I at the Homebrew Computer Club in 1976. For \$666.66, buyers received a blank printed circuit board, a parts kit, and sixteen-page assembly manual. The power supply, keyboard, storage system, and display were not included.

Company in Palo Alto. Inspired by the Altair, Woz started building his own computer based on the 6502 microprocessor produced by MOS Technology, Inc. It was the cheapest fully functional microprocessor at the time, substantially undercutting the price of Intel’s 8080. In six months, Woz had produced a circuit board for the 6502, with 4 kilobytes of memory and circuitry that allowed it to be directly connected to a monitor and keyboard. This was a great improvement in usability compared to toggling the switches on the Altair. He unsuccessfully tried to interest Hewlett-Packard, his employer, in commercializing it, but received an enthusiastic reception at the Homebrew Computer Club.

In 1971, a friend had introduced Woz to teenager Steve Jobs, a fellow computer enthusiast (B.8.4). Together, Woz and Jobs designed and sold “blue boxes,” unauthorized devices that enabled purchasers to mimic the control signals of the Bell Telephone Company’s lines and make calls for free. After high school, Jobs went to Reed College in Portland, Oregon, but dropped out of full-time education and returned home to Los Altos, California. He went to work for Atari Inc., one of the first video game companies, until he had saved enough money to visit India to pursue his interest in Asian philosophy. When Jobs returned from India in 1974 he immediately saw the potential in Woz’s personal computer board. Together with Ronald Wayne, Jobs and Wozniak founded the Apple Computer Company on 1 April 1976 to market the board that Woz had designed as a personal computer kit, later called the Apple I (Fig. 8.9). Wayne later sold his shares back to Jobs and Wozniak. Jobs persuaded the newly established Byte Shop store to order one hundred boards at \$500 each. To get the funds to buy the chips and have the circuit boards manufactured, Jobs had to sell his Volkswagen van and Wozniak his programmable Hewlett-Packard calculator! They assembled the boards in the garage of Jobs’s parents’ home in Los Altos, and eventually managed to sell about two hundred computer kits and make a small profit. Jobs realized that the microprocessor-based computer could appeal to a much broader market than just computer enthusiasts if it was presented as a self-contained machine in a plastic case,



B.8.4. Steve Jobs (right) and Steve Wozniak met in a friend’s garage in the late 1960s. The two of them bonded over their shared interest in electronics and practical jokes. Their first project together was to design, produce, and sell “blue boxes” that enabled the user to make long-distance telephone calls for free.



Fig. 8.10. When the Apple II was released in 1977, it was promoted as “an extraordinary computer for ordinary people.”⁹ The self-contained system, user-friendly design, and graphical display made Apple a leader in the first decade of personal computing. Unlike the earlier Apple I, for which users had to supply essential parts such as a case and power supply, the Apple II was intended to be a fully realized consumer product. Apple’s marketing emphasized its simplicity as an everyday tool for home, work, or school.

with a standard power supply, a keyboard and screen, and a cassette tape for long-term storage of data and programs. In addition, the computer would need a high-level programming interface and, potentially, a range of application software, including video games.

With this specification from Jobs, Woz set about creating the Apple II while Jobs set about getting the plastic cases made and raising start-up money (Fig. 8.10). Woz’s design for the Apple II is recognized as a masterpiece of circuit design. It used fewer chips than the Altair, had good color graphics, and was great for the interactive games that Woz loved to play. Taking a page from the Altair playbook, Woz argued strenuously for the use of an open bus architecture with slots for expansion so that other companies could expand the machine’s capabilities in interesting ways. He also wrote a version of BASIC for the machine. Meanwhile, Jobs had been introduced to Mike Markkula, only thirty-four at the time but already able to retire from his job as Intel’s marketing manager with a considerable fortune generated by his Intel stock options. Markkula recognized the potential of the two young entrepreneurs and bought a third of the company and helped them write a business plan and raise venture capital. The Apple II was a great success and the advertising campaign claimed:

The home computer that’s ready to work, play and grow with you.... You’ll be able to organize, index and store data on household finances, income taxes, recipes, your biorhythms, balance your checking account, even control your home environment.¹⁰

In reality, of course, there was no software to monitor your biorhythms, balance your checkbook, or perform any of these household applications at the time; most of the software available was still only for playing games.

For application software to really take off the personal computer needed a better and more convenient storage medium. Cassette tapes were slow and awkward, and could not provide random access; a user had to scroll through the tape from the beginning to reach any given point. These inconveniences disappeared with the invention of the *floppy disk* by David Noble of IBM in 1971. Floppy disks were flexible plastic disks coated with magnetic material that could be used to store information. IBM introduced the initial eight-inch floppies for loading the microcode for its mainframe computers. Alan Shugart, a former IBM manager whose team had helped develop the floppy disk, realized that this technology would be the ideal memory device for personal computers and set up a company to manufacture 5½-inch floppy disks and disk drives. Although Apple purchased the drives from Shugart, Woz thought that the controlling circuits were too complex, requiring as many as fifty chips in total for their implementation. In another engineering *tour de force*, Woz redesigned the disk drive controller using only five chips and was able to deliver a floppy disk drive controller for the Apple II that was both simple and fast.

In 1979 the first “killer” business application for the personal computer emerged –an application that the Xerox PARC team had missed. This was the spreadsheet, a table used to present financial and other information. The first spreadsheet program was VisiCalc (Fig. 8.11), short for Visible Calculator. It was the brainchild of Daniel Bricklin (B.8.5), a twenty-six-year-old Harvard MBA

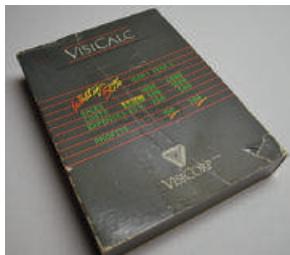


Fig. 8.11. The 1979 program, VisiCalc, was the first “killer” application for business. It was a spreadsheet program produced by Daniel Bricklin and Bob Frankston for their Software Arts company. Many customers bought an Apple computer specifically to run VisiCalc. Although VisiCalc was the first spreadsheet for personal computers, it was soon followed by other spreadsheet programs from Lotus, Microsoft, Borland, and others and eventually lost its supremacy in the market. Bricklin had not been able to patent the spreadsheet idea in VisiCalc because software patents were not generally issued until after a groundbreaking U.S. Supreme Court ruling in 1981.

student who had seen his fellow MBA students struggle to perform tedious and error-prone arithmetic operations on rows and columns of financial data. He conceived of VisiCalc as a program that would automate these spreadsheet calculations, and with Robert Frankston, who had worked with him on Project MAC at MIT, he set up a company to develop and market his new VisiCalc application. Although Apple was not interested in marketing the program directly, VisiCalc rapidly became a word-of-mouth success. As Robert Slater says in his book *Portraits in Silicon*:

Suddenly it became obvious to businessmen that they had to have a personal computer: VisiCalc made it feasible to use one. No prior technical training was needed to use the spreadsheet program. Once, both hardware and software were for hobbyists, the personal computer a mysterious toy, used if anything for playing games. But after VisiCalc the computer was recognized as a crucial tool.¹¹

Apple was incorporated in January 1977. When it went public in December 1980, it was one of the most successful stock offerings in Wall Street history and Jobs and Wozniak became multimillionaires overnight.

Project Chess and the IBM PC

By 1980, IBM had observed the rise of the Apple II and other microprocessor-based computers. A small group of advocates within the company realized that IBM could only become a dominant player in this emerging personal computer market if it could produce a machine very quickly. According to the



B.8.5. Bob Frankston (left) and Dan Bricklin, pioneers of the VisiCalc spreadsheet. Bricklin graduated from MIT in 1973 with a degree in electrical engineering and computer science. After some years in industry, he signed up for an MBA from Harvard Business School. It was while Bricklin was sitting in room 108 in Aldrich Hall at Harvard in 1978 that he dreamed of an easier way to calculate financial projections for multiple different business scenarios: “Imagine if my calculator had a ball in its back, like a mouse....”^{B1} He wrote a first prototype for the Apple II, which introduced rows and columns and some arithmetic operations. With fellow MIT graduate Bob Frankston, Bricklin founded Software Arts, Inc. in 1979 and began selling VisiCalc for \$100 a copy. There is a plaque on the wall of Aldrich 108 commemorating Bricklin’s achievement: “In this room in 1978, Dan Bricklin, MBA ’79 conceived of the first spreadsheet program. VisiCalc, original ‘killer App’ of the information age, forever changed how people use computers in business.”^{B2}

IBM archives, Bill Lowe and Don Estridge (B.8.6) of the IBM lab in Boca Raton, Florida, suggested the timescale required:

One analyst was quoted as saying that “IBM bringing out a personal computer would be like teaching an elephant to tap dance.” During a meeting with top executives in New York, Lowe claimed his group could develop a small new computer within a year. The response: “You’re on. Come back in two weeks with a proposal.”¹²

It was a controversial decision for IBM to enter the personal computer business. One insider was even reported as saying:

Why on earth would you care about the personal computer? It has nothing at all to do with office automation. It isn’t a product for big companies that use “real” computers. Besides, nothing much may come of this and all it can do is cause embarrassment to IBM, because, in my opinion, we don’t belong in the personal computer business to begin with.¹³

The two most important decisions made by Frank Cary, IBM’s chairman and CEO, were not only that the development of an IBM personal computer or PC should go ahead, but also that its development could proceed outside of IBM’s normal processes. In particular, the Boca Raton team was free to build the system using a non-IBM microprocessor, and they chose to use the new 16-bit Intel 8088 chip. An 8-bit microprocessor, as used in the first generation of personal computers, could only access 8 bits of data in a single machine instruction. The next generation microprocessors like Intel’s 8088 could access and process 16 bits at a time. In a further significant break from IBM’s standard practices, Lowe also had permission to outsource the software to vendor companies. A 1979 business study undertaken for IBM evaluating the prospects for microprocessor-based computers had advised the company not to develop proprietary systems and applications because “in order to succeed IBM would need a lot of third parties writing software for the new system.”¹⁴ The conclusion was clear: IBM would purchase an operating system from an outside company. This decision also implied that the vendor of the operating system could put its software on non-IBM machines.

Jack Sams was the IBM engineer in charge of software development for the PC prototype. In the summer of 1980, Sams led a delegation from IBM to Microsoft’s offices in Seattle, where they briefed Allen and Gates about their top-secret effort to build an IBM personal computer, code-named Project Chess. According to IBM historian Edward Bride:

Sams met with Bill Gates to evaluate whether Microsoft could handle the task of writing a BASIC compiler for the IBM PC. This led to his recommendation to William Lowe that they use Microsoft software in the final product. In addition, when he was unable to make a deal with Intergalactic Digital Research for the operating system, Sams and his team turned to Microsoft. This led to the development of an operating system released by IBM as PC-DOS and by Microsoft as MS-DOS.¹⁵

Microsoft agreed to supply compilers not only for BASIC but also for FORTRAN, COBOL, and Pascal, all delivered on IBM’s tight timetable.



B.8.6. Don Estridge (1937–85) led Project Chess – the top-secret project to develop an IBM PC at its Boca Raton plant in Florida. In an unprecedented move by IBM, the machine had an open architecture and used third-party hardware and software. Estridge died in a plane crash three years after the PC’s introduction – by then the PC was a runaway success and IBM had sold more than a million machines.

Intergalactic Digital Research, later shortened to Digital Research, was a company set up by Gary Kildall (B.8.7) to market his CP/M software, an operating system for microprocessors, including personal computers. CP/M was then the leading *disk operating system* (DOS) for computers with one or more disk drives. *Dr. Dobb's Journal*, a magazine aimed at computer programmers, had announced CP/M to hobbyists in 1976 as being similar to DECSYSTEM 10 in that it used commands derived from DEC's operating system software. For example, it specified a disk drive by a letter; file names had a period and a three-character extension; and the "DIR" command enabled the user to see the available files in a directory. In 1977, Kildall had rewritten CP/M so that only a small part of the software needed to be customized for each new machine. He called this specialized code the BIOS, for Basic Input/Output System. The BIOS standardized personal computer system software in the same way that the Altair bus had standardized the hardware.

For reasons that are still unclear, the IBM delegation decided they could not reach an agreement with Digital Research for CP/M and came back to Microsoft. Their return presented Microsoft with a dilemma because the company was not at that time in the business of writing operating system software. Concerned that the whole deal with IBM might now be in jeopardy, Allen and Gates looked around for alternatives. A local company called Seattle Computer Products (SCP) was producing 8086 16-bit hardware, and a designer from SCP, Tim Paterson, had been working with Paul Allen on testing his prototype hardware using Allen's 8086 BASIC software. As an interim measure while waiting for Gary Kildall to deliver his long-promised 16-bit version of CP/M, Paterson had also developed a program he called QDOS, standing for Quick and Dirty Operating System. Paul Allen and Gates then made a deal with Rod Brock, the owner of SCP, for Microsoft to license QDOS, now renamed 86-DOS. In July 1981, Microsoft went back to Brock and negotiated the outright purchase of all rights to 86-DOS. This deal was probably the best value in the history of computing and provided the foundation for Microsoft's future success.

The IBM Boca Raton team had committed to delivering a hardware prototype to Microsoft "before December 1" of 1980. It was actually delivered early in the morning of Monday, 1 December. Microsoft's business manager, Steve Ballmer, answered the door and showed the IBM team to a small, windowless backroom, which was kept under lock and key, with access limited to only a handful of people. Difficulties with the unreliable hardware caused problems for Microsoft's software teams, and they missed the original mid-January deadline for both PC-DOS and BASIC. In the end, the IBM PC, IBM's personal computer, was announced in August 1981 and shipped ahead of schedule in November (Fig. 8.12). Besides PC-DOS from Microsoft, there were two other operating systems available - CP/M-86 from Digital Research and p-System from the University of California, San Diego. As David Bradley said, "Simple economics determined the winner - PC-DOS sold for about \$40, while CP/M-86 and p-System were about \$400."¹⁶ The IBM planners had estimated that "in the five-year lifetime of the IBM PC, sales from all sources would equal 241,683 units."¹⁷ The corporate staff at IBM actually scolded the planners for suggesting such unrealistically large sales volumes. In fact, according to Bradley: "Over the



B.8.7. Gary Kildall (1942–94) had a PhD in computer science from the University of Washington and was teaching at the Naval Postgraduate School in Monterey, California, when he developed the first commercially successful operating system for microcomputers – Control Program for Microcomputers or CP/M in 1974. He and his wife then established a company – Digital Research – to market CP/M. IBM approached Kildall about providing CP/M for its PC project but for reasons that remain obscure, Kildall and IBM were unable to reach an agreement. IBM then went back to Microsoft, who then created their phenomenally successful PC-DOS operating system.

The Computing Universe



Fig. 8.12. The IBM PC. IBM's first personal computer arrived in 1981, more than five years after personal microcomputers first arrived. However, the IBM name instantly legitimized the business market and gave companies the confidence to invest in personal computers for word processing and spreadsheet work. Although IBM had introduced the PC in 1981 with an advertising campaign aimed at the general public, the IBM PC had its most profound impact in the corporate world. Companies bought PCs in bulk, revolutionizing the role of computers in the office – and introducing MS-DOS to a vast user community. Unlike most previous IBM products, the PC incorporated hardware and software from other companies. The PC also had an open architecture, which allowed a thriving “PC clone” business to develop.



Fig. 8.13. An IBM PC button featuring a Charlie Chaplin-like figure.

PC's five-year lifetime, IBM sold approximately 3 million systems, 250,000 in one month alone in 1984.”¹⁸

During the next few years, the IBM PC became an industry standard, and most of the popular software packages were converted to run on the machine (Fig. 8.13). In January 1983, the editors of *Time* magazine nominated the IBM PC as their “Man of the Year.” The openness of the architecture and the standardization of the operating system software encouraged other manufacturers to produce *IBM-compatible computers*, also called *IBM clones*, which copied the features of the IBM PC. IBM remained the technology leader and produced several very successful successors to the original PC, most notably its second-generation personal computer, the PC AT – the letters AT stood for *advanced technology*. The PC AT bus allowed expansion by the easy insertion of printed circuit boards. In 1987 IBM tried to introduce some proprietary technology into the PC market with the Personal System/2 or PS/2 computer, replacing the now-standard but limited 16-bit PC AT bus with the more capable Micro Channel Architecture. Although IBM was willing to license the technology to others, the strategy to regain a proprietary advantage was not a success. Eventually the PC AT bus was superseded by the Peripheral Component Interconnect (PCI) interface, an architecture created by an industry consortium in 1993. As Mark Dean, a participant in the original IBM PC design team, now says:

I'd have to admit that we lost sight of why the PC had become successful when we went to the PS/2. To enable continued growth, we should have continued with the model of building it so that other people can play. That would have allowed us to stay in control of the market. When we did the PS/2, we lost control.¹⁹

The Macintosh and Microsoft Windows

For all of the creativity at Xerox PARC in the 1970s, the success of personal computing – first with the Apple II and then with the IBM PC – owed nothing to any of their research. This situation changed in 1979 when Steve Jobs was invited to visit PARC. At the insistence of Xerox higher management, PARC showed Jobs its Alto-based vision of the office of the future. Larry Tesler remembers Jobs asking, “Why isn't Xerox marketing this? ... You could blow everybody away!”²⁰ In fact, microprocessor technology was not yet powerful enough to support all the features he had seen. When Xerox released the Xerox Star in 1981, it was not a commercial success despite wonderful reviews and its many advanced features, such as the capability to network the computer to a



Fig. 8.14. The Macintosh computer was announced in 1984 in a now-famous advertisement during the U.S. Super Bowl football game. The video was made by Ridley Scott and contrasted the regimented world of IBM's PC dominance with the creativity made possible by the Macintosh with explicit reference to Big Brother and George Orwell's novel 1984.

laser printer and other Star machines. The Star was too expensive (more than \$10,000) to compete with the “good enough” approach of the IBM PC released later that year. As a result of his visit to PARC, Jobs recruited Larry Tesler to lead the development of a new Apple computer to be called the Lisa, named after Jobs’s daughter. The Lisa was launched in 1983, but, at a price point of nearly \$10,000 like the Star, it was not a commercial success. However, there was another new Apple computer in the works.

The Macintosh project had been started in mid-1979 by Jef Raskin, who had been a professor of computer science at the University of California in San Diego. He was familiar with the work at PARC, and he wanted to produce a machine with a built-in screen that was so simple and easy to use that a user could just plug it in and get started right away. The machine was called the Macintosh, after Raskin’s favorite apple. When Jobs returned from PARC, he took over the project. Raskin had wanted to produce a machine for less than \$1,000. At Jobs’s insistence, Apple added new PARC-like features including a mouse and this increased the price. The Macintosh finally went on sale in 1984 for nearly \$2,500 (Fig. 8.14). To build the hardware and the software for the Macintosh, Jobs isolated the design group in a separate building over which a pirate’s flag was hoisted. John Sculley, later CEO of Apple, said:

Steve’s “pirates” were a hand-picked band of the most brilliant mavericks inside and outside Apple. Their mission, as one would boldly describe it, was



B.8.8. Steve Jobs (1955–2011) was a university dropout who played a key role in shaping today’s computing universe. With the talented engineer, Steve Wozniak, Jobs founded Apple Computer in 1976 to market the Apple I personal computer kit. The Apple II was released in 1977 as a self-contained consumer product that was great for playing games but also ran the VisiCalc spreadsheet software, the first killer application for business. After a famous visit to Xerox PARC in 1979 at which Jobs saw the Alto personal computer and its GUI, Apple produced the revolutionary Macintosh computer in 1984.

After falling out with the Apple Board and CEO John Sculley, Jobs was effectively fired from Apple in 1985 and sold all but one of his shares. He then founded the NeXT computer company and its first computer workstation was released in 1990 – and used by Tim Berners-Lee, at CERN in Geneva, to develop the World Wide Web. NeXT reported its first profit of just more than \$1 million in 1994.

In 1986, Jobs bought a 70 percent stake in a graphics company later called Pixar that helped Disney computerize its ageing animation department. Pixar’s digital animation business was originally just a sideline to their hardware and software business. Jobs was losing money at both NeXT and Pixar but all this changed in 1995 with the success of Pixar’s full-length animated movie *Toy Story*, with Jobs credited as executive producer.

In 1996, Apple had lost market share dramatically, and Jobs was invited back to Apple as an adviser with an agreement that Apple would buy NeXT for around \$400 million. By 1997, Jobs had the title interim CEO, inevitably abbreviated as iCEO. In the first year that Jobs came back, he laid off more than three thousand employees and Apple lost more than \$1 billion in 1997. After two years of huge losses, Apple had returned a \$300 million profit by 1998. As CEO, Jobs oversaw a succession of phenomenal successes – starting with the iMac, followed by the iTunes store, the iPod, the iPhone, and the iPad. With its touch interface, Jobs completely reinvented the mobile phone as can be seen at a glance by the number of people using touch phones in every situation. In 2003, Jobs was diagnosed with cancer of the pancreas and although an initial treatment had some success, his health declined and he died in October 2011.



Getting comfortable with an IBM PC now can put you in a good position later.

Familiar with a 1200 Personal Computer can give you a real advantage in school. And it can put you in position when you get out. Because the skills accepted throughout the education system—proficiency with a keyboard, the ability to type quickly, choices now you'll be making a lot of decisions—will be useful when you leave school. In knowledge is one thing you can take with you. And put to good use.

You can count on the IBM to help. You can depend on its power. Its memory is large enough to store powerful programs that can meet your academic, present and future needs. And for the convenience you get out, the IBM has a built-in monitor and keyboard. It runs on standard electrical power, so you don't have to worry about batteries.

The IBM's word processing and graphic capabilities—coupled with an IBM mouse—make it easy to produce polished and professional. And every time you print a paper or plot a graph, you'll know it's done right.

The IBM's word processing and graphic capabilities—coupled with an IBM mouse—make it easy to produce polished and professional. And every time you print a paper or plot a graph, you'll know it's done right.

For more information, see your computer supplier or write to: IBM Personal Computer products, Dept. P.O. Box 1000, Somers, NY 10589, or your local IBM Authorized PC Dealer.

Fig. 8.15. IBM personal computer advertisement. Marketing computers to students was a new experience for IBM. They had certainly never before suggested to customers that they could use them "under your favorite tree."

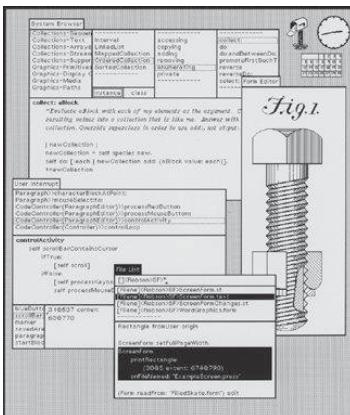


Fig. 8.16. Screenshot of the Cedar environment running on a Xerox Alto. The GUI had windows, icons, menus, and a pointing device – the WIMP interface we still use today. Steve Jobs was inspired by his glimpse of the Alto's GUI that he saw at Xerox PARC. Jobs was sure that this was the way forward for personal computing and he committed Apple to this approach.

to blow people's minds and overturn standards. United by the Zen slogan "The journey is the reward," the pirates ransacked the company for ideas, parts, and design plans.²¹

Unlike the Lisa, which achieved excellent performance by using more expensive specialized hardware, the Macintosh attempted to provide Lisa-like features using the commodity Motorola 68000 microprocessor. Jobs obsessively oversaw every aspect of the Macintosh's design, even including the colors of the production facilities and the designers of the case, and his name appears on the design patent. One of the designers, Terry Oyama, later said: "Even though Steve didn't draw any of the lines, his ideas and inspiration made the design what it is. To be honest, we didn't know what it meant for a computer to be 'friendly' until Steve told us."²² Jobs got each of the forty-seven people from the original Macintosh design team to sign their names inside the molding; these original Macs are now collectors' items.

In spite of the Mac's impressive capabilities, it was not successful as a consumer product, and the lack of an option to incorporate a hard disk meant that it could not displace IBM from the business market (Fig. 8.15). It did gain a loyal following in the publishing and media industries, where it came to the fore because of its powerful capabilities for *desktop publishing*, in which editors and designers used computers to edit text and lay out pages. Unlike the Apple II and the IBM PC, the Macintosh architecture was closed, and third parties were not able to add circuit boards offering additional functionality. Although Microsoft supplied some application software, Apple had developed its own operating system and it was difficult for developers to write applications that made optimal use of the hardware.

Users liked the exciting look and feel of the Macintosh's *graphical user interface* – abbreviated as GUI and pronounced “gooey.” This was the revolutionary system developed by researchers at Xerox PARC that enabled users to give instructions to the computer through a *WIMP interface* – standing for windows, icons, menus and pointers. The use of windows here is not a reference to Microsoft's operating system of the same name, but to a rectangular frame called a *window* that appears on the computer screen. A window can run a program at the same time as other windows on the same screen are running other programs. The user can see the output from all the programs on the screen and can enter information into any program by selecting the corresponding window. Icons are small pictures representing specific actions that the user can select. A *menu* is a list of available options, typically shown by icons and a *drop-down menu*, which lists programs or applications when selected by the user. Lastly, a *pointer* is a marker, such as an arrow, that appears on the screen to allow the user to select an operation. For a long time the most common way of controlling the pointer has been with a *mouse*, a palm-sized device that enabled the user to move an arrow on the screen and to select icons of drop-down menus by clicking a button.

The success of the Mac GUI made it clear that the next important step would be the development of a similarly powerful interface for the IBM PC and its clones (Fig. 8.16). Several companies, including Digital Research and IBM, attempted to produce a similar interface for the PC. Microsoft had begun

work on a GUI project after Bill Gates had visited Jobs at Apple and had seen the prototype Macintosh computer in development. The Microsoft product was originally going to be called “Interface Manager,” but Scott McGregor, who had joined Microsoft from PARC, had written the window manager component for PARC’s interactive programming environment and had called his PARC software “Windows.” Rowland Hanson, the head of marketing, then persuaded Gates to call Microsoft’s new operating system “Windows,” as Hanson explained, “to have our name basically define the generic.”²³ Version 1 of Windows appeared in 1985. The highest performance Intel microprocessor for the PC at the time was the 80286, called the “286” for short, but even on this chip the Windows GUI ran far too slowly. It was only when the Intel 386 and 486 chips became available in the late 1980s that using windows on Windows really became a practical proposition. Meanwhile, the company had also developed a new GUI-based operating system called OS/2 with IBM, released in 1987. But by early 1989, Microsoft had sold some two million copies of Windows and OS/2 was history. When Windows 3.0 launched in May 1990, Bill Gates (B.8.10) was finally able to say that it “puts the ‘personal’ back into millions of MS-DOS-based computers.”²⁴ However, it was not until the release of Windows 3.1 in 1992 that the original PARC vision of computing for the masses truly arrived.

During the 1980s, Microsoft had been developing application software for the Macintosh and, in so doing, had learned how to develop software for a windows-based interface. When Microsoft designers applied this experience to the PC, Gates followed the example of Steve Jobs in insisting that each application adhere to a common GUI. With Charles Simonyi (B.8.9) having left Xerox PARC and now at Microsoft developing Word for Windows, and with the Excel spreadsheet program, first developed for the Mac, Microsoft could finally put these together with the PowerPoint presentation software to form an “office suite.” By a wholehearted commitment to GUIs, and by bundling three applications together as Office, Microsoft was finally able to overtake the PC market leaders for word processing and spreadsheets, WordPerfect and Lotus 1-2-3.

Meanwhile, Microsoft’s lawyers were battling a lawsuit filed by Apple charging that Microsoft had infringed “the Company’s registered audio-visual copyrights protecting the Macintosh user interface.”²⁵ After four years of legal



B.8.9. Charles Simonyi wrote Bravo, the first WYSIWYG word processor, while at Xerox PARC. He later “took the PARC virus” to Microsoft where he was responsible for creating the hugely successful Word for Windows application. Simonyi also helped develop a system of programming that allowed Microsoft to manage increasingly complex software projects involving large teams of programmers. The style involved a systematic way of naming variables – called “Hungarian” because of its apparent incomprehensibility. Simonyi has used some of his personal fortune from his time at Microsoft to become an astronaut – as shown here – and he has visited the International Space Station on two occasions.



B.8.10. Bill Gates is one of the best-known faces of the personal computer revolution and it would be hard to find a person who would not recognize his name. At the age of thirteen, he was enrolled in Lakeside School, an exclusive preparatory school in Seattle. When Gates was in eighth grade the school purchased an ASR-33 teletype and some computer time for students on a GE computer and he wrote his first BASIC programs. With Paul Allen and some other Lakeside students, he was allowed free use of a DEC PDP-10 computer at the nearby offices of the Computer Center Corporation provided they assisted in debugging the operating system software. At age seventeen, Gates and Allen formed their first joint venture called Traf-O-Data for making hardware and software for automating a traffic counting system. The enterprise was not a success but Gates and Allen developed valuable experience and a powerful set of tools for the PDP-10. Gates graduated from Lakeside in 1973 and enrolled at Harvard. Publication of the January 1975 issue of *Popular Electronics* stimulated Allen and Gates to develop a basic interpreter for the Altair 8800 computer. In 1976 they established Microsoft Corporation to develop software for the growing microcomputer market. Microsoft's partnership with IBM to develop the MS-DOS operating system for the IBM PC was a critical step for the company. Bill Gates had a remarkable vision for Microsoft: "a personal computer on every desk and in every home." With the advent of the Internet and the World Wide Web, in 1995 Gates turned the company around to embrace the Web with his famous "The Internet is a tidal wave" memo. In 2006 Bill Gates transitioned out of his day-to-day involvement with Microsoft and now devotes a significant amount of his time to philanthropic activities with the Bill and Melinda Gates Foundation (B.8.11). With fellow billionaire Warren Buffett, Gates champions the cause of "creative capitalism" – a combination of capitalism and philanthropy to solve some of the urgent problems facing the world.

arguments, a federal judge dismissed Apple's lawsuit in 1992, ruling that "Apple cannot get patent-like protection for the idea of a graphical user interface, or the idea of a desktop metaphor [under copyright law]...."²⁶ In Walter Isaacson's biography of Steve Jobs, Bill Gates is quoted as ending an angry meeting with Jobs by saying: "Well, Steve, I think there's more than one way of looking at it. I think it's more like we both had this rich neighbor named Xerox and I broke into his house to steal the TV set and found out that you had already stolen it."²⁷

A post-PC era?

With the progress of Moore's law, the scale of computers has been extended from large "mainframe" business computers to microprocessor-based personal computers. From Osborne's first portable computer – which was more "luggable" than truly portable – we now have smart phones and tablets that are rapidly changing the way we interact with computers. These are more than just new "form factors" for the PC where the term *form factor* refers to the size,



B.8.11. Melinda and Bill Gates visiting with mothers taking part in a malaria intervention treatment program at the Manhiça Health Research Center in Mozambique. Bill and Melinda announced in Manhiça that their foundation was awarding three grants totaling \$168 million to fight malaria. The grants will accelerate the search for a malaria vaccine, new drugs to fight drug-resistant malaria, and new treatment strategies for children.

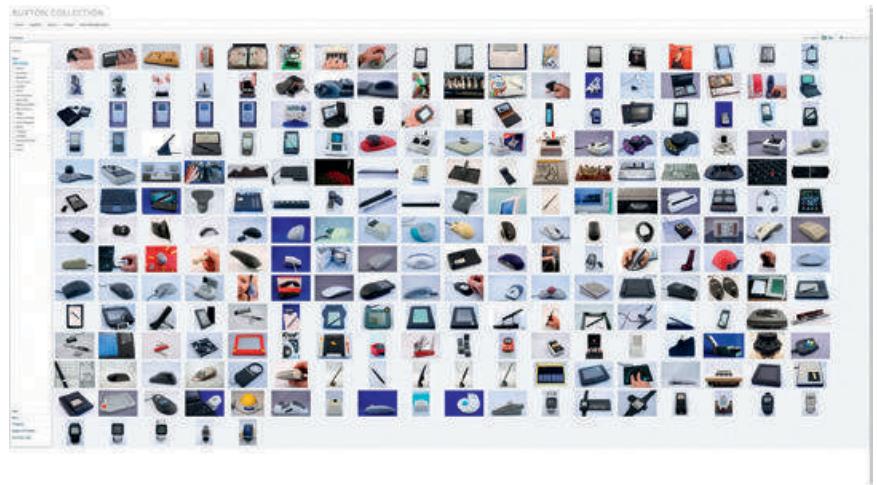


Fig. 8.17. Bill Buxton is a Canadian computer scientist and designer and a pioneer in the research field of human-computer interactions. The Buxton Collection is a collection of interactive devices that he has been collecting for about thirty-five years. The collection is intended as a resource for those interested in design, user experience, and the history of interaction. Buxton is a principal researcher at Microsoft Research and an expert on birch bark canoes.

configuration, and physical arrangement of a hardware device. Because these devices can now be with us all the time new modes of interaction are threatening to displace the mouse as our primary interaction mechanism (Fig. 8.17).

The term *smart phone* was introduced by Ericsson in 1997. A smart phone is just a mobile phone that uses a microprocessor-based computing platform to provide the computing power we expect from a PC. The first smart phone was the “IBM Simon” introduced in 1993. In addition to being a mobile phone it served as a *personal digital assistant*, providing a calendar, address book, calculator, notepad, and clock. Simon ran a version of DOS and could also play games but only a few thousand were sold. In 2002, the Canadian telecommunications company Research in Motion (RIM) introduced its first BlackBerry smart phone, which combined the ability to send and receive email with the capabilities of a mobile phone. We will explore the emergence of the Internet, email, and the World Wide Web in [Chapters 10](#) and [11](#). Easy access to email and the web, together with the increasing availability of “Wi-Fi” allowing wireless connectivity to the Internet, have been two of the key drivers for the emergence of new portable computing and communication devices such as smart phones and tablet computers.

Many companies have tried to market smart phones, tablets, and personal digital assistants with a variety of different user interfaces. The history of touch screen input goes back a long way. The first touch screen using capacitive technology was invented by E. A. Johnson at the Royal Radar Establishment in Malvern, United Kingdom, in the 1960s for an air-traffic control application. The device works by sensing the change in electric charge caused by a finger touching the screen. Another common type of touch technology used on Point of Sale systems is based on the change in resistance caused by pressing on a flexible surface. Other mechanisms for interacting with computers include



Fig. 8.18. The first “Newton” message pad from Apple was released in 1993. This handheld device used an ARM RISC processor and ran various applications for handwriting recognition, note taking, sketching, Internet access, and other productivity tools. The more sophisticated iPhone incorporates many features of its predecessor.

voice input, handwriting with a stylus, and gesture recognition. However, it was not until Steve Jobs unveiled the iPhone in 2007 that there has been mass adoption of such devices by consumers (Fig. 8.18). Instead of a mouse or a stylus, the iPhone used touch as its input mechanism and transformed the user experience.

Jobs and Apple also created an “App Store” where third-party vendors could market their applications for the iPhone. By 2012, more than half a million applications were available for download – copied electronically to the user’s own iPhone. In terms of usage, in the first two months of 2012, more than half of user sessions on iPhones were spent playing games! From being a frivolous application wasting valuable computer time, Moore’s law has transformed the economics so that computer games are now seen as immensely valuable. It is this theme that we take up in [Chapter 9](#).

Key concepts

- Bus
- 16-bit microprocessor
- Graphical user interface
- Desktop metaphor
- WIMP Interface: windows, icons, menus, and pointers
- Mouse
- Bitmap
- Touch screen input



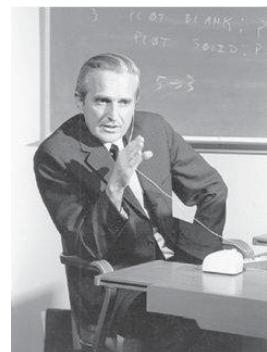
Three pioneers of interactive computing

Licklider and human-computer symbiosis

Although his name is not well known to the general public, few people have been more influential in the evolution of computing than J. C. R. Licklider (see Chapter 10, B.10.1, for a brief biography). Lick, as he asked everyone to call him, studied psychology as an undergraduate at Washington University in St. Louis and followed up with a PhD in psychoacoustics at the University of Rochester. After a spell at Harvard, Lick moved to MIT in 1950. His interest in information technology and human-computer interactions had led him to become involved with the Semi-Automated Ground Environment (SAGE) air-defense system. The Cold War threat of an airborne nuclear attack on the United States had triggered the development of SAGE. The system derived from Jay Forrester's original Whirlwind project at MIT. Forrester's team built a prototype for the SAGE system, and IBM, the lead contractor for the project, produced the system for installation. Each SAGE computer could track up to four hundred airplanes and support up to fifty terminals. From his experience with this project, Lick had become convinced of the value of computers to analyze data in real time, as the data came in, instead of being confined to the traditional batch processing of mainframe computers. Lick summarized his ideas in an influential paper called "Man-Computer Symbiosis," published in 1960, arguing for the need to develop computers that could interact with humans to support real-time decisions – that is, decisions that need to be made at the actual time that events were happening. In 1962, Lick was given a unique opportunity to implement his vision for the future of computing. He was invited to lead a new research program at the U.S. Department of Defense's Advanced Research Projects Agency (ARPA). Lick was based in the Pentagon and had, as Michael Hiltzik says, "one 'cloak-and-dagger' project under his nominal jurisdiction [that was] so highly classified even he was not cleared to know what it was."²⁸ Licklider's Information Processing Techniques Office (IPTO) ultimately had a larger computing research budget than all of the other U.S. government agencies combined. Lick's strategy was to place his trust in a small number of talented individuals and outstanding centers of academic excellence. Lick nurtured interactive computing research not only at MIT but also at the University of California in Berkeley, Carnegie Mellon University, Stanford University, and the University of Utah. He gave researchers significant amounts of funding that enabled them to pursue long-term research goals without too much interference or frequent proposal writing. His program of interactive computing ultimately delivered major advances in many important areas, including networking, computer graphics, software engineering, and human-computer interactions. A major component of his program was a \$3 million grant to MIT for Project MAC (Project on Mathematics and Computation), a pioneering time-sharing system that eventually could support up to thirty users at any one time. Lick's ARPA centers at Utah and Stanford generated almost all of the ideas embodied in today's computer user interfaces. David Evans and Ivan Sutherland headed the graphics research group of at the University of Utah, and Doug Engelbart (see B.8.12) led the Human Factors Research Center at the Stanford Research Institute.

One of Lick's most significant contributions while at ARPA was his role in helping establish computer science as a valid research discipline in universities. Bob Taylor, one of his successors at ARPA, said:

Prior to his [Lick's] work at ARPA, no U.S. university granted a Ph.D. in computer science. A university graduate program requires a research base, and that in turn requires a long-term commitment of dollars. Lick's



B.8.12. Doug Engelbart (1925–2013) was best known as the inventor of the mouse. In fact Engelbart played a leading role in many major developments in computing. This photo shows him rehearsing for the demo in 1968, which has entered computing history as the "Mother of All Demos."

ARPA program set the precedent for providing the research base at four of the first universities to establish graduate programs in computer science: U.C. Berkeley, CMU, MIT, and Stanford. These programs, started in 1965, have remained the country's strongest and have served as role models for other departments that followed. Their success would have been impossible without the foundation put in place by Lick in 1962–64.²⁹

Doug Engelbart and the mouse

On Engelbart's return home from military service in World War II, he had been inspired by reading Vannevar Bush's visionary essay "As We May Think," published in 1945. Bush accurately foresaw the days when scientists would be drowning in information:

Publication has been extended far beyond our present ability to make real use of the record. The summation of human experience is being expanded at a prodigious rate, and the means we use for threading through the consequent maze to the momentarily important item is the same as was used in the days of square-rigged ships.³⁰

Bush envisioned a machine he called the "memex": "a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility."³¹ In 1957, when Engelbart joined the Stanford Research Institute (SRI), he was finally able to start realizing his dream of creating a memex with funding initially from Bob Taylor, then at NASA, and later from Licklider at ARPA. (See Chapter 10, B.10.10 for a brief biography of Taylor.)

Engelbart and his team are best known for their invention of the mouse (Fig. 8.19), but they also pioneered many other features of the present-day GUI, in which the user controls a cursor on the screen to select options from menus, start programs by clicking icons, and perform other operations. Engelbart was not sure why it was called a mouse: "None of us would have thought that the name would have stayed with it out into the world, but the thing that none of us would have believed either was how long it would take for it to find its way out there."³²

Engelbart's researcher, Bill English, created the first mouse out of a hollowed-out block of wood with two small wheels that allowed the user to control the movement of a cursor on the computer screen (Fig. 8.20). At an event that has been called the "Mother of All Demos" at a major computing conference in San Francisco in December 1968, Engelbart demonstrated his group's "electronic office" software, called NLS (short for oN Line System), in which he introduced the mouse, video conferencing, word processing, a real-time editor, and split-screen displays to the world. He also demonstrated a prototype of Bush's memex idea by showing how the user could select a single word in a text document and be instantly linked to a second document. This prototype was the first implementation of a *hypertext* system, which enables the user to jump from one document to another, such as we now use daily on the World Wide Web. Butler Lampson and Peter Deutsch, both early recruits to Xerox PARC, had worked part-time for Engelbart in the 1960s and were both influenced by the vision of the NLS electronic office software and by the 1968 demo.

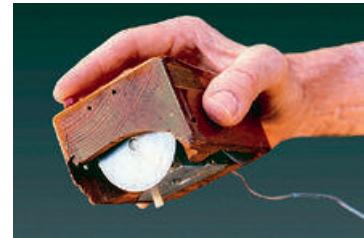


Fig. 8.19. Doug Engelbart's mouse from 1967. This prototype mouse, invented by Engelbart at the Stanford Research Institute, rolled on two sharp wheels facing 90 degrees from each other.

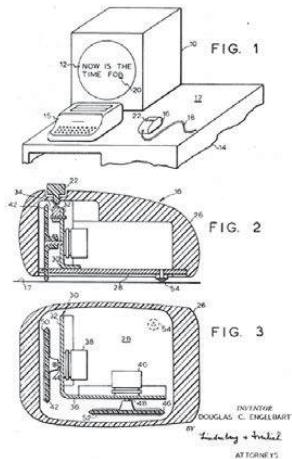


Fig. 8.20. Engelbart's "mouse" patent drawings. The word *mouse* does not appear in Engelbart's patent for the computer pointing device. The knife-edged wheels each rolled in just one direction, transmitting movement information for that direction. Each slid without turning when the mouse was moved in the other direction.

Bob Taylor and Xerox PARC

Bob Taylor (see B.10.10) had a master's degree in psychology and was working for NASA as a project manager when he was invited to the Pentagon to meet Licklider. The two scientists had a shared background in psychoacoustics but also shared a vision for the future of interactive computing. When Licklider left ARPA in 1964, he persuaded Ivan Sutherland, creator of Sketchpad, one of the first interactive graphics programs, to leave MIT and take on management of the IPTO program with Taylor as his deputy. Sutherland stayed only a short time so Taylor soon found himself running the entire IPTO program. He continued to support the embryonic U.S. computer science community and a vision of interactive and networked computing. He organized annual IPTO research conferences and gained an unrivaled personal knowledge and the trust of the most creative individuals in the U.S. computing research community. This served him well when he was appointed to recruit researchers for the Computer Science Laboratory at Xerox's brand new Palo Alto Research Center (PARC) that had just opened in June 1970.

PARC was the inspiration of Xerox's CEO, Peter McCollough, who realized that the copier market would soon become much more competitive with the expiration of one of Xerox's key patents. McCollough wanted Xerox to own what he called the "office of the future." The new mission for Xerox was to control "the architecture of information." Taylor brought together a cast of computing superstars at PARC. These stars included Butler Lampson and Chuck Thacker, both rescued from a failed Berkeley start-up called the Berkeley Computer Corporation, and Alan Kay, one of Ivan Sutherland's research students from the University of Utah. Kay's vision was to build a "Dynabook" – "a notebook-shaped machine with a display screen and a keyboard you could use to create, edit, and store a very personal sort of literature, music, and art,"³³ – exactly the sort of vision that Taylor and Licklider had wanted their IPTO program to generate. One more element of the mix at PARC came from another of their investments, Engelbart's Augmentation Research Center at the nearby Stanford Research Institute.

Taylor wanted to incorporate elements of Engelbart's vision of interactive computing into PARC, so he recruited Bill English, the engineer who had done the detailed design work for the mouse. Eventually others from Engelbart's team followed, and the ideas of the Augmentation Research Center team and their NLS electronic office system became essential elements of PARC's own vision for interactive computing. At PARC, the stage was set. Taylor believed that, having provided the researchers with the overall vision and funding, his job was now to keep out of the way and let them do what they did best.

Taylor was a key player in the history of Xerox PARC (Fig. 8.21) but was also a controversial figure. Nevertheless, in his resignation speech he could fairly say:

Most people spend a lifetime without opportunities for pioneering completely new ways of thinking about large collections of ideas. I have been fortunate to have been a leader in three: time-sharing; long-distance interactive networking; and personal distributed computing.³⁴



Fig. 8.21. Lab Director Bob Taylor held periodic informal meetings in the "beanbag" conference room where his Xerox PARC staff presented their new technical ideas. Speakers always received frank and honest feedback from their colleagues.

From minicomputers to portable computers

The rise and fall of DEC

Although users had been able to run their own programs on early computers like the ENIAC and the EDSAC, a new usage model developed as the market for commercial computers exploded in the 1950s. A new occupation called *computer operator* emerged. Computers sat in air-conditioned machine rooms, and computer operators kept the machines running and loaded jobs onto the computer instead of allowing users direct access to the hardware. This system was called *batch processing*, and although it made efficient use of the very expensive computer, it could also be extremely frustrating for users trying to debug their programs. Typically, a program consisted of a stack of cards on which the user had punched holes to represent data. The computer operators fed the cards into the computer, and the user received a printout for the program some hours later, or even the next day if the program was run overnight. As you can imagine, users found it extremely tiresome to wait twelve hours for their program to run. Sometimes, they picked up the printout only to find an error message saying that the program could not run because the computer had received an incorrect instruction, which might be as minor as a mistyped comma in one of the program statements. As we saw in Chapter 3, such frustration led John McCarthy to develop the idea of time sharing, in which multiple users could be connected to the computer simultaneously and have the illusion that they were the sole user. The computer switched its attention from user to user, executing a small part of each user's program during each slice of time. Because even the early computers could perform many thousands of operations each second, users had the impression that their program was running all the time. Users interacted with the machine using their own computer *terminal*, typically just a combination of keyboard and screen, but they could also input programs and data using much faster paper tape or punch card readers.

The efforts of McCarthy and others at MIT achieved success in 1961 when Fernando Corbató introduced the Compatible Time-Sharing System (CTSS), one of the first working time-sharing systems, running on the MIT Computation Center's IBM computers. IBM was skeptical at first, but the success of time sharing in the early 1960s led to the establishment of new companies that offered commercial time-sharing services. Customers would buy time on the expensive machines and pay for it by the minute. For a few brief years, time sharing seemed like the path to the future, but it was soon overtaken by the development of the minicomputer. In 1957, Ken Olsen (B.8.13), an electrical engineer from MIT, had the bold idea of starting a new type of computer company. Olsen had worked with Jay Forrester on Project Whirlwind to develop a computer for the U.S. Navy. In 1952, while still a graduate student, Olsen and his fellow graduate student Harlan Anderson had played a major role in building a machine called the Memory Test Computer to try out Forrester's ideas on magnetic core memories, working with one of the MIT computing pioneers, Wesley Clark.

What was Olsen's bold idea? Frustrated with the slow development of interactive and time-sharing computing at MIT, he decided that



B.8.13. Ken Olsen (1926–2011) began his career by fixing radios in his basement in Bridgeport, Connecticut. As a graduate student at MIT, he and fellow student Harlan Anderson built the Memory Test Computer to evaluate the feasibility of using magnetic core memory. While at MIT's Lincoln Labs, they were responsible, with Wesley Clark, for designing and building the first transistorized research computer, the TX-0. With his brother and Harlan Anderson, Olsen founded the first minicomputer company, the Digital Equipment Corporation. Their first computer, the PDP-1 was based on the TX-0. From 1957 until 1992, Digital's headquarters was located in a former wool mill in Maynard, Massachusetts.

there was a market for small, inexpensive machines he called *minicomputers*. Many of the computational problems required by the business and the research community were actually relatively small, such as calculating a payroll or monitoring an experiment. So in 1957, Olsen, his colleague Harlan Anderson, and Ken's younger brother Stan decided to go into the computer business for themselves. With several thousand dollars of their own money, supplemented by funds from a Boston investment firm, they set up the Digital Equipment Corporation – also known as DEC and later just as Digital – in a Civil War-era wool mill just outside Boston, Massachusetts. Three years later, they produced their first computer, the Programmed Data Processor model 1, commonly known as the PDP-1. This machine cost \$120,000 and provided much more cost-effective computing than was then available from IBM and others. DEC's business really took off with the introduction of the PDP-8 in 1965, generally regarded as the first minicomputer (Fig. 8.22). The PDP-8 machine used transistors and magnetic core memory and cost \$18,000. It could only run one program at a time and had less memory than a mainframe computer, but it became the first commercially successful minicomputer. The key selling point was its price and ability to be easily coupled with laboratory instruments for experimentation and control. Because of the low cost of the PDP-8, many more customers could afford to buy their own computer to do their routine computational tasks. As computer historian Stan Augarten reports in his 1984 book *Bit by Bit*:

Scientists ordered PDP-8s for their laboratories; engineers got them for their offices; the Navy installed them on submarines. In refineries, PDP-8s controlled the flow of chemicals; in factories, they operated the machine tools; in warehouses, they kept track of inventory; in computing centers, they ran programs that didn't require the power of a mainframe; in banks, they kept track of accounts. The notion of the information utility gave way to *distributed processing*. For example, a bank would install a minicomputer in each of its branches; the machines handled the branches' transactions during the day and sent records of their transactions to the bank's central computer at closing time. The applications were endless.³⁵

While Bob Metcalf was a graduate student at MIT and Harvard, DEC lent him a PDP-8. It was stolen from his lab and he had no idea how he could repay DEC. However, the company took the news in its stride and ran an advertisement for the PDP-8 as "the first computer small enough to steal."³⁶

In 1965 DEC pursued a second path for low-cost, interactive computing by introducing the PDP-6 as the first commercial time-sharing system. This used many of the concepts and functions of MIT's CTSS software and also DEC's experience with a time-shared PDP-1 at BBN, specially designed for Licklider. Introduced in 1965 and the forerunner of the PDP-10, the PDP-6 was a 36-bit, time-shared mainframe computer with roughly the same power as the IBM 709X and 110X series mainframe batch computers. Thus DEC grew rapidly, beginning in the mid-1960s and through the 1970s along two paths: classical minicomputers like the PDP-8 and PDP-11 (introduced in 1970), and the PDP-10 time-shared computers that could support one hundred or more active users and were used by universities and by time-sharing service companies. The PDP-10 ran a time-sharing system called TOPS-10. The Computer Center Corporation or "C-Cubed" installed one of the first PDP-10s in the Seattle area in 1968. To help debug the system, the company offered free time on the computer to a couple of local teenagers named Paul Allen and Bill Gates. In the early 1980s, the two paths were covered by the PDP-11 minicomputers that used single chip microprocessors, and the VAX-11 computers that were typically time shared and could be used in clusters. By 1980, almost one hundred companies had started building minicomputers using integrated circuits. By 1985, only six of these companies remained.



Fig. 8.22. A PDP-8 on a tractor used for controlling sowing.

The architecture of the PDP-11 popularized the idea of a *bus*, a set of connections linking all the major components of the machine, including the central processor, memory, and I/O devices, in a standard way. The bus architecture was important because it allowed both DEC and “original equipment manufacturers” – often known as OEMs – to easily add extra units and to customize the machine for specialized applications. By the mid-1970s, the minicomputer market had become very competitive and DEC needed to offer a computer with more memory than the PDP-11, a 16-bit computer that could access 64 kilobytes of memory. The VAX 11/780, announced in October 1977, was the first commercially available 32-bit computer. It supported 2^{32} or 4 gigabytes of virtual address space. *Virtual memory* is a mechanism for swapping data in and out of a small, fast main memory from a slower, larger memory on a *hard disk*, a rigid magnetic disk permanently mounted in the computer’s disk drive and used to store data. The VAX had sixteen 32-bit registers and could understand a large, complex *instruction set* (set of commands). The computer architect Gordon Bell (B.8.14) led the initial design effort for the VAX, as head of DEC’s R&D organization. Bell had served as the architect of many of DEC’s successful machines. The VAX turned out to be a runaway success. It offered cost-effective high performance compared to the much more expensive mainframe computers. It also had a user-friendly operating system called VMS and came with a standard set of languages and library software (Fig. 8.23).

Despite these early successes that allowed DEC to become the second-largest computer company through the 1980s, DEC no longer exists. Although it built the foundation for inexpensive interactive computing, it missed out on the personal computer revolution. The IBM PC was announced in 1981 and used Intel’s 8088 microprocessor, which had all the essential features of a computer on a single chip. So why was DEC not able to succeed in this new market? Although DEC’s founder, Ken Olsen, is often quoted as saying, “There is no reason for any individual to have a computer in his home,”³⁷ DEC really tried hard to succeed with personal computers. In 1982, DEC introduced three incompatible personal computers – the DECMate, based

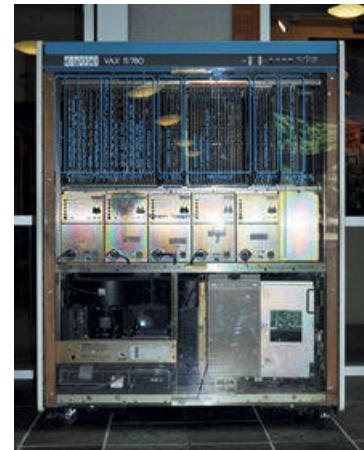


Fig. 8.23. DEC’s very successful VAX-11/780 computer was the result of a small architecture team of six engineers led by Gordon Bell, including Dave Cutler, who was responsible for the VMS software architecture and its implementation. The VMS operating system was much liked by users but universities often put up a version of the UNIX operating system as an alternative. Both Bell and Cutler later went on to work for Microsoft.



B.8.14. Gordon Bell grew up in Missouri and helped in the family electrical business, repairing appliances and wiring homes. He graduated from MIT with a degree in electrical engineering and then spent time in Australia programming an English Electric DEUCE computer, a production version of Alan Turing’s Pilot ACE computer. In 1960, Bell was recruited by DEC where he worked on the early PDP machines and designed the first UART – Universal Asynchronous Receiver/Transmitter – that converted bytes of data and transmitted the individual bits sequentially. As a faculty member at Carnegie Mellon University, with Allen Newell, he introduced the processor-memory-switch (PMS) and instruction-set processor (ISP) notations for describing computer structure and architecture. When back at DEC in the 1970s as the head of their R&D organization, he led the design team that developed the enormously successful VAX computer. In the 1980s, Bell was the founding Assistant Director of the U.S. National Science Foundation’s Computer and Information Science and Engineering Directorate. In 1997 he established the Gordon Bell Prize for outstanding achievement in high performance computing applications. In 1995 he joined Microsoft where he built a version of the memex of Vannevar Bush. His recent work on the MyLifeBits project aims to capture digitally all the significant events each day in a person’s life including geographical locations, conversations, phone calls, messages sent, and even the web pages visited.

on the PDP-8 and sold as a word processing machine; the DEC Professional, a more powerful machine than the IBM PC but based on the PDP-11 architecture and a proprietary bus; and the DEC Rainbow, an “almost IBM PC compatible” platform. DEC engineers prided themselves on their expertise in computer architecture and “refused to be part of the pack and compete with others by supplying competitive but fully compatible machines.”³⁸

Nevertheless, it is too simplistic to attribute DEC’s demise solely to its failure in the PC market. Although it is true that DEC had made several bad management decisions, in the 1990s, with the rise of the Internet and the web (see Chapters 10 and 11), the company was still well placed to become a market leader for Internet products. DEC had extensive expertise in networking and servers, and also had pioneered one of the first successful web search engines with their AltaVista offering. In the end, Gordon Bell, Vice President of Engineering at DEC during the 1970s, believes that “Failure was simply ignorance and incompetence on the part of DEC’s top 3–5 leaders and, to some degree, its ineffective board of directors that in removing Olsen made an even worse mistake in appointing [his replacement] Palmer.”³⁹

The time machine: The Alto

In 1972, Chuck Thacker (B.8.15), Butler Lampson (B.8.16), and Alan Kay (B.8.17) at Xerox PARC conceived of building a revolutionary new type of computer. Instead of batch processing or time sharing on a mainframe or minicomputer, the “Alto” was intended to be a genuine personal computer small enough to fit under a desk (Fig. 8.24). To computer designers and businesses at the time, computers were expensive devices. Just to provide the computer memory for a single-user machine would cost many thousands of dollars. “But to Thacker and his colleagues such objections missed the point,” Hiltzik says, and explains:



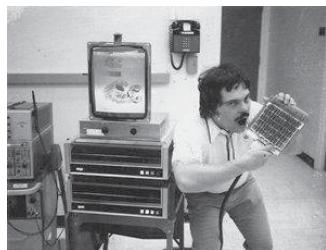
B.8.15. Charles (Chuck) Thacker (left) and Butler Lampson at Xerox PARC. Thacker is a Turing Award recipient and designer of Xerox PARC’s Alto computer – the first truly personal computer. He had learned from his experience at the unsuccessful Berkeley Computer Corporation that in designing computing systems, “less is often better than more.” Thacker’s word for describing engineering projects that had got out of hand was “biggerism” – as in “This project has been biggered.” Thacker also had an influence in introducing the WYSIWYG capability in the Bravo word processor. When his wife Karen was typing a paper for a class, he suggested she try using an early version of the Bravo word processor. She commented that she needed to see what she got in print on the screen. Thacker passed this comment to his colleagues at PARC and Bravo was soon able to do this.



B.8.16. Butler Lampson was the software architect of Xerox PARC’s famous Alto personal computing system. At PARC, he also made major contributions to the first WYSIWYG word processor, Ethernet for local area networking, operating systems, and laser printers. Lampson received the ACM Turing Award in 1992 and the remarkable citation read: “For contributions to the development of distributed, personal computing environments and the technology for their implementation: workstations, networks, operating systems, programming systems, displays, security, and document publishing.”⁴⁰ His wife, Lois Lampson, was the first person to produce her PhD thesis on a laser printer. When she submitted her thesis, the administrator insisted on knowing which was the original – to be deposited in the library – and which was the copy! This photo was taken at the Rome NATO software engineering conference in 1969.



Fig. 8.24. The Xerox PARC Alto, which featured a mouse, removable data storage, networking hardware, a visual user interface, easy-to-use graphics software, and email. The Bravo and Gypsy word-processing software offered the user the first “What-You-See-Is-What-You-Get” or WYSIWYG interface, with printed documents matching what users saw on screen. The Alto for the first time combined these and other now-familiar elements in one small computer. Developed by Xerox as a research system, the Alto marked a radical leap in the evolution of how people interact with computers, leading the way to the environments we still have on today’s computers. By making human-computer interactions more intuitive and user friendly, the Alto opened computing to a much wider range of users, from experts to nonspecialists, including children. People were able to focus on using the computer as a tool to accomplish a task rather than on learning their computer’s technical details. When it was built, the revolutionary Alto would have been a very expensive personal computer if put on sale commercially. Lead engineer Charles Thacker noted that the first Alto probably cost in the region of \$12,000 to build: as a product, the price tag might have been as much as \$40,000. A decade later, Moore’s law had reduced costs and personal computers with adequate memory became affordable.



B.8.17. Alan Kay’s name is closely linked with the development of personal computing. He started college but left before graduation to join the air force. In the air force he found a new interest in computing and when he left he enrolled at the University of Colorado. Kay graduated in 1966 with degrees in mathematics and molecular biology and went on to graduate work at the University of Utah where he obtained an MS in electrical engineering and a PhD in computer science in 1969. It was at Utah that Kay conceived of the Dynabook – a portable, personal computer rather like the iPad of today, but he could not create it with the technology of the time. He joined Xerox PARC in 1971 and his research team created the overlapping windowing GUI interface. Kay was also one of the creators of the Smalltalk programming language and coined the name *Object Oriented Programming*. The picture on the screen is the first animated bitmapped graphic: the Sesame Street cookie monster eating a cookie. Kay received the 2003 Turing Award for pioneering object-oriented programming and “for fundamental contributions to personal computing.”

The Alto aimed to be not a machine of its time, but of the future. Computer memory was horrifically expensive at the moment, true, but it was getting cheaper every week. At the rate prices were falling, the same memory that cost ten grand in 1973 would be available in 1983 for thirty dollars. The governing principle of PARC was that the place existed to give their employer that ten-year head start on the future. They even contrived a shorthand phrase to explain the concept. The Alto, they said, was a time machine.⁴⁰

Thacker, Lampson, and Kay all agreed that they needed to build a fast, compact machine with a *high-resolution* display – that is, a display with images that were sharp and finely detailed. For Alan Kay, it would not be a complete realization of his vision but it would at least be, as he said, an “interim Dynabook.”⁴¹ Thacker began designing the machine in November 1972, and the first prototype was up and running in an incredibly short time by April 1973. One of the major challenges was powering a high-resolution display without using unreasonable amounts of processor power and memory. Thacker’s solution was to use a *bitmap*, a representation of an image consisting of rows and columns of dots. Each bit in the computer’s memory corresponds to a dot or *pixel* on the display screen. This bitmap had been inspired by experiments in Kay’s group that used a block of memory that normally stored custom fonts to display images. The screen had a resolution of 606 by 808 pixels. This meant that nearly half a million bits needed to be refreshed thirty times a second, which was a great deal of processing power and memory for the time. As processing power and computer memory became cheaper, these limitations rapidly disappeared, as predicted by Lampson. The Alto was the future.

Despite the excitement generated by the Alto, Lampson was only too well aware that it still needed real application software to be useful. Lampson was sketching the requirements for a text-editing program when Charles Simonyi (B.8.9) walked into his office. Simonyi had been an undergraduate at Berkeley when

Lampson was a graduate student and they had collaborated on the CAL TSS time-sharing system. He had also worked with Lampson at the ill-fated Berkeley Computer Corporation while still a student. After working for a while on a parallel computing project called Illiac-IV, he rejoined his ex-Berkeley colleagues at PARC. Lampson supplied Simonyi with three sheets of notes capturing his thoughts for an interactive text editor. Simonyi called his new word processing system "Bravo." Using the Alto's bitmapped screen, he was able to encode complex typefaces, boldface, italic, and underlining in the text together with a detailed page layout, so that the document appeared on the screen almost exactly as it would be printed out. Bravo was thus the first WYSIWYG word processor – What You See Is What You Get – and it became a great hit among the engineers at PARC. As Simonyi later said:

It was the killer app, no question. People would come into PARC at night to write all kinds of stuff, sending letters, doing all personal correspondence, PTA reports, silly little newsletters, anything. If you went around and looked at what the Altos were doing, they were all in Bravo.⁴²

In spite of its popularity with the engineers at PARC, Bravo needed a more user-friendly interface if it was to be adopted by the much larger community of nonengineers. Lampson and Thacker had made a deliberate decision not to work on the user interface of Bravo, not because they did not think it was important but because they did not have the resources to do both the implementation and the user interface. It was left to Bob Taylor to initiate such a project with two other computer scientists at PARC, Larry Tesler and Tim Mott.

Before joining PARC, Tesler had produced a program called Pub that helped ordinary users format and print their documents. At PARC, Tesler had been a member of a team trying to reengineer and update a version of Engelbart's interactive multimedia system. He rapidly became dissatisfied with the complexity of the system being created and was eager to take on a new challenge. Tim Mott was an Englishman with a computer science degree from the University of Manchester who was working in the United States for a Xerox subsidiary called Ginn & Company that published textbooks. Determined to try to get some value from Xerox's "corporate research" tax, Mott's boss Darwin Newton sent Mott to visit PARC and see how their research on office systems could assist him as a publisher. Mott concluded that their system was much too complex and difficult for the publishing company to use: "There wasn't a lot of time spent looking at what mere mortals would be able to do with the system."⁴³ Taylor challenged Mott to use the Alto to produce something useful.

Tesler and Mott also found the user interface of Bravo far too complicated. It was usable by experts but not easy or attractive for ordinary users like publishers. Mott went back to Ginn & Company and did some market research on what nonengineers actually wanted from such a program. Unsurprisingly, he found that the publishers wanted the program to mimic what they actually did with their paper-based process. This is the origin of the "cut" and "paste" commands that are still used to this day. Tesler and Mott called their new system Gypsy, and it was the first program to use the mouse to execute point-and-click operations in the way we do today.

While Simonyi, Tesler, and Mott were developing Bravo and Gypsy, Alan Kay's group at PARC was still pursuing his Dynabook dream. The Alto's bitmapped screen allowed enormous flexibility in what could be displayed. So why can't a user write a memo in one part of the screen and use a drawing program in another part? This led the team to think of the screen in terms of a "desktop" metaphor where electronic documents could be piled on top of one another, just like papers on a desktop. They created overlapping boxes, or "windows," for each different task. But shifting these boxes around put a huge demand on the Alto's processor and was extremely slow. In a stroke of genius, Dan Ingalls came up with "BitBlt," an abbreviation of "bit boundary block transfer" and pronounced *bitblit*. Instead of having the computer change each of the components of a rectangular image individually for the new location, BitBlt operated on the entire bitmap using fast Boolean operations to create the new image. This new technique meant that the user could rapidly scroll up or down the text of a document on the screen by moving a mouse. It also meant that windows could be created and moved around at will, and that the illusion of a stack of papers on a desk could be

convincingly executed. When Kay's team was demonstrating their system to their skeptical engineering colleagues, Ingalls stunned the audience by using a mouse click to call up a drop-down menu listing several possible commands and selecting the "cut" command. As Hiltzik says:

The PARC user interface, with its overlapping windows, mouse clicks, and pop-up menus, had entered computing history. More than twenty-five years and many engineering generations later, it remains the indisputable parent of the desktop metaphor guiding the users of millions of home and office computers.⁴⁴

Bob Taylor's leadership was the key ingredient in PARC's astonishing success. According to Butler Lampson:

The master often speaks in somewhat inscrutable fashion with a deeper and more profound interpretation than his humble disciples are able to provide. In retrospect you can really see that the path has been plotted years in advance, and you've been following his footsteps all along.⁴⁵

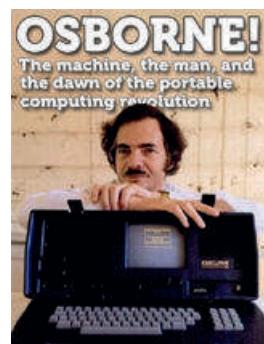
Chuck Thacker agreed: "As a leader of engineers and scientists he had no equal. If you're looking for the magic, it was him."⁴⁶

The Osborne portable computer

At the beginning of the 1980s, many small companies were entering the personal computer market with creative ideas. In July 1981, a British computer designer named Adam Osborne (B.8.18) launched a portable personal computer that became a big success with traveling business executives. One of the advantages was that it was designed like a briefcase that could fit under the airplane seat. The computer was based on the popular Z80 microprocessor designed by Federico Faggin, who had designed the first Intel microprocessor back in 1971. The Osborne 1 (Fig. 8.25) had two floppy disk drives; 64 kilobytes of memory; a five-inch, fifty-two-column scrollable display; and a modem connection that could send and receive data by telephone. The reason for the small display was portability: a larger display could be easily damaged during transportation. The Osborne 1 ran the CP/M operating system and its bundled software included a BASIC interpreter, the Word-Star word processing software, and a SuperCalc spreadsheet program. The price of the computer was very attractive and generated a huge demand. The Osborne Computer Corporation grew from two employees to three thousand within a year. However, Osborne Computer made some critical mistakes that caused its sudden decline in a fiercely competitive market. The company declared bankruptcy in 1983. Adam Osborne described the reasons for this demise in his book *Hypergrowth: The Rise and Fall of Osborne Computer Corporation*.



Fig. 8.25. The first "Portable Computer" – the Osborne 1 – was released in 1981. It weighed 24.5 pounds (12 kg) and cost US\$1795 – just more than half the cost of a computer from other manufacturers with comparable features – and ran the CP/M operating system. It was designed to fit under an airline seat. At its peak, the Osborne Computer Corporation was shipping ten thousand units per month.



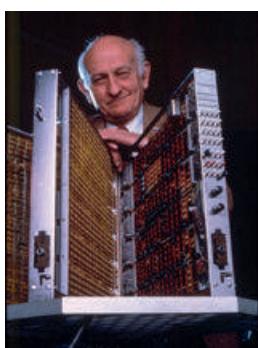
B.8.18. Adam Osborne (1939–2003) was best known for creating the first commercially available portable computer, the Osborne 1, released in April 1981. The Osborne 1 included both word processing and spreadsheet software and the company was briefly very successful. The Osborne 1 was produced at a time when IBM did not bundle hardware and software with their PCs: customers had to buy the operating system software and the monitor separately.

Developments in computer architecture: RISC and ARM

The idea of RISC – standing for Reduced Instruction Set Computing – originated on the East and West coasts of the United States at around the same time in the early 1980s. At IBM Research in the 1970s and 1980s, John Cocke (B.8.19) had investigated how often the individual instructions of an instruction set were actually executed when running a representative set of programs. He discovered that a small set of instructions occurs more frequently than others and proposed that only this reduced instruction set should be implemented in hardware. The more complex instructions of the standard approach can then be built up out of this smaller set. Having only a small instruction set simplifies the circuit design and enables us to build fast computers with low power consumption.

On the West Coast, David Patterson at Berkeley and John Hennessy at Stanford were pursuing similar ideas. It was Patterson who coined the name RISC, for *reduced instruction set computing* architecture – in contrast to the usual *complex instruction set computing* (CISC) architecture of a standard microprocessor. The first RISC processor was introduced in an experimental research computer called the IBM 801 in 1980. Cocke's ideas made their way into the IBM POWER architecture – an acronym for Performance Optimization With Enhanced RISC. This led to the introduction of IBM's RS/6000 (RS for RISC System) workstations in 1990. Cocke's colleague, Fran Allen, worked with him on the interaction of computer architectures and compilers and they were responsible for developing many innovative compiler optimization techniques (see B.8.20). In recent years there has been a coming together of RISC and CISC. The new microprocessors of Intel's x86 series externally support a CISC instruction set of almost nine hundred instructions, but internally only a RISC subset of instructions are actually implemented in silicon.

For smart phones and tablets, power consumption and battery life is very important. The United Kingdom-based company ARM Holdings – ARM standing for Advanced RISC Machines – had its origins in the Acorn computer company. In the United Kingdom, Acorn had great success with a personal computer called the BBC Microcomputer (Fig. 8.26). In looking for a microprocessor for their next generation machine, they took the unusual step of deciding to design their own microprocessor. Herman Hauser, the CEO of Acorn, encouraged two of his engineers, Steve Furber and Sophie Wilson, to look at the Berkeley RISC papers and then sent them on a fact finding visit to the United States. They visited Bill Mensch, CEO of the Western Design Center in Phoenix, Arizona, and were amazed at the tiny scale of his globally successful operation. As Wilson tells it: “A couple of senior engineers, and a bunch of college kids ... were designing this thing....



B.8.19. John Cocke (1925–2002) received a BS degree in mechanical engineering from Duke University in 1946 and later went back to Duke to complete a PhD in mathematics in 1956. He then joined IBM Research where he remained for the rest of his working life. At a symposium in honor of John Cocke in 1990, Fred Brooks described him as a “fire starter” because of his constant stream of ideas: “The metaphor that comes to mind is of a man running through a forest with flint and steel, striking sparks everywhere.”¹⁵ After working on IBM’s Stretch project, an ambitious effort to build the fastest scientific computer, and the Advanced Computer Systems research project, in 1975 Cocke led the research team building the experimental IBM 801 computer, which pioneered the ideas of RISC architectures and optimizing compiler technology. In the 1980s, these ideas led to the IBM POWER architecture and the RS/6000 RISC workstations. In 1987, Cocke received the Turing Award for the development of RISC and for his work on optimizing compilers with Fran Allen. In his 1990 talk, Fred Brooks characterized Shannon, von Neumann, and Aitken as the three “greats” of the first generation of computer scientists; and Knuth, Sutherland, and Cocke as the three “greats” of the next generation.



B.8.20. Fran Allen grew up on a farm in New York State and obtained an MSc degree in mathematics from the University of Michigan. She joined IBM in 1957 initially to earn enough money to pay off her school loans and stayed at IBM for the next forty-five years. Her first assignment was to teach IBM research scientists about the new FORTRAN language that had been developed by John Backus. This set her interest to compilers and she collaborated with John Cocke on the interaction of computer architecture and compilers in both the IBM Stretch and ACS projects. In 2006, Allen became the first woman to receive the Turing Award for her “pioneering contributions to the theory and practice of optimizing compiler techniques that laid the foundation for modern optimizing compilers and automatic parallel execution.”^{F7}

We left that building utterly convinced that designing processors was simple.^{F47} They came back to Cambridge thinking: “Well, if they can design a microprocessor, so can we.”^{F48} Eighteen months later in April 1985, they had a working ARM chip.

ARM is different from other microprocessor companies such as Intel and AMD in that it licenses its technology to other companies rather than having its own manufacturing facilities. The strategy has been spectacularly successful. ARM technology is now used in 95 percent of smart phones and 80 percent of digital cameras and, by 2012, more than forty billion ARM-based chips had been shipped.

Bell's law for the birth and death of computer classes

In 1972, Gordon Bell observed that, with Moore’s law predicting that the number of transistors would double every eighteen months, and with the introduction of Intel’s 4004 microprocessor in 1971, it was possible to predict the broad outlines of the next forty years of computer evolution. He suggested that there would be two evolutionary paths for computers: (1) evolution at constant price with increasing performance; and (2) evolution at constant performance but with decreasing cost. Bell’s law states that, roughly every decade, a new, lower-priced computer class forms based on a new programming platform that results in new usage patterns and the establishment of a new industry. Examples of such new classes are the emergence of the minicomputer, the personal computer, and the smart phone (Fig. 8.27).



Fig. 8.26. The BBC Microcomputer. The British Broadcasting Company’s Computer Literacy Project in the early 1980s hoped “to introduce interested adults to the world of computers.”^{F6} Acorn produced this popular computer in 1981 so viewers at home could emulate what they saw demonstrated on *The Computer Programme* TV series.

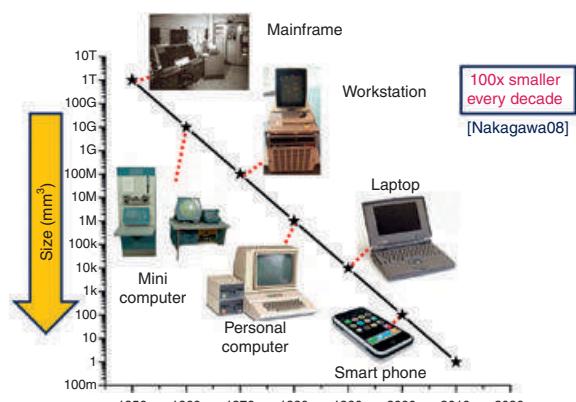


Fig. 8.27. A pictorial explanation of Bell’s law.

Odds and ends

The origins of Control-Alt-Delete

One of the most enduring features of personal computers that survives to this day was the brainchild of IBM engineer David Bradley. He was part of the original twelve-person IBM PC hardware design team and was responsible for developing the program to get the computer system started after you turn it on. The development team needed an easy way to reset and reboot the machine. The simplest solution would have been to include a reset button in the hardware, but it turned out that the mechanical construction of the PC made this method difficult. Bradley then decided to use the keyboard and deliberately chose a set of three keys that would be difficult to hit by accident: “Control” (Ctrl) and “Alt” were on the left side of the keyboard, while the “Delete” (Del) key was on the far right (Fig. 8.28). The Control-Alt-Delete combination for reset was originally intended only as a development tool, but the command soon entered into general use. Because IBM had a deadline and the reset command was just one of many problems to solve, Bradley said, “After 10 minutes of design, coding and testing [of Control-Alt-Delete], it was time to move on to the next problem.”⁴⁹



Fig. 8.28. Ctrl-Alt-Del cushions for programmers to sleep better.

Did QDOS copy CP/M?

One question that is sometimes raised is how similar Paterson’s “Quick and Dirty Operating System” QDOS was to CP/M. In fact, Paterson had not had access to the source code of CP/M, and he had written QDOS using the CP/M user’s manual and Intel’s documentation for the 8086 microprocessor. Like CP/M, QDOS used the DEC system commands “Type,” “Rename,” and “Erase,” and also retained Kildall’s idea of how to make it convenient to modify and customize the computer system by using a BIOS for input and output. One of the DEC commands in CP/M that was changed in QDOS and MS-DOS was the all-purpose PIP command, where PIP stands for Peripheral Interface Program, a method used for transferring files between devices. This command was replaced by the much less cryptic “Copy” instruction. QDOS also introduced a more efficient file system for storing floppies by using Microsoft’s File Allocation Table or “FAT” file system, which is still widely used today.

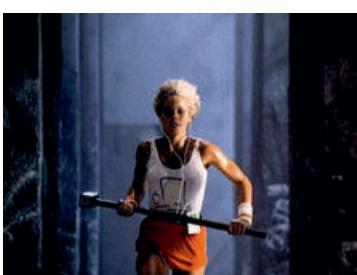


Fig. 8.29. The sledgehammer is about to smash the screen with its “Big Brother” image in Apple’s famous 1984 Super Bowl advertisement for the Macintosh. The commercial was directed by Ridley Scott and produced two years after the release of his dystopian science fiction movie *Blade Runner*. In 1995 it was ranked as one of the “50 best” television commercials ever made by the editors of the trade publication *Advertising Age*.

The Mac and Big Brother

The Macintosh was introduced to the world in a legendary Super Bowl television advertisement in January 1984 that subtly linked the dominance of the IBM PC to 1984, George Orwell’s novel of a dystopian future (Fig. 8.29). *Hard Drive*, a history of Microsoft by James Wallace and Jim Erickson, describes the ad:

[The commercial] showed a roomful of gaunt, zombie-like workers with shaved heads, dressed in pajamas like those worn by concentration camp prisoners, watching a huge viewing screen as Big Brother intoned about the great accomplishments of the computer age. The scene was stark, in dull, gray tones. Suddenly, a tanned and beautiful young woman wearing bright red track clothes sprinted into the room and hurled a sledgehammer into the screen, which exploded into blackness. Then a message appeared: “On January 24, Apple Computer will introduce the Macintosh. And you’ll see why 1984 won’t be like 1984.”⁵⁰

9 Computer games

Video games are bad for you? That's what they said about rock and roll.

Shigeru Miyamoto¹

The first computer games

Since the earliest days, computers have been used for serious purposes and for fun. When computing resources were scarce and expensive, using computers for games was frowned upon and was typically an illicit occupation of graduate students late at night. Yet from these first clandestine experiments, computer video games are now big business. In 2012, global video game sales grew by more than 10 percent to more than \$65 billion. In the United States, a 2011 survey found that more than 90 percent of children aged between two and seventeen played video games. In addition, the Entertainment Software Association in the United States estimated that 40 percent of all game players are now women and that women over the age of eighteen make up a third of the total game-playing population. In this chapter we take a look at how this multibillion-dollar industry began and how video games have evolved from male-dominated "shoot 'em up" arcade games to more family-friendly casual games on smart phones and tablets.

One of the first computer games was written for the EDSAC computer at Cambridge University in 1952. Graduate student Alexander Douglas used a computer game as an illustration for his PhD dissertation on human-computer interaction. The game was based on the game called tic-tac-toe in the United States and noughts and crosses in the United Kingdom. Although Douglas did not name his game, computer historian Martin Campbell-Kelly saved the game in a file called OXO for his simulator program, and this name now seems to have escaped into the wild. The player competed against the computer, and output was programmed to appear on the computer's cathode ray tube (CRT) as a display screen. The source code was short and, predictably, the computer could play a perfect game of tic-tac-toe (Fig. 9.1).

Like OXO on the EDSAC, most of the early computer games ran on university mainframe computers and were developed by individuals in their spare time. The game Spacewar! (Figs. 9.2a and 9.2b) was one of the earliest



Fig. 9.1. Alexander (Sandy) Douglas created a version of tic-tac-toe or noughts and crosses for the Cambridge EDSAC computer in 1952. The output was displayed on a CRT screen.

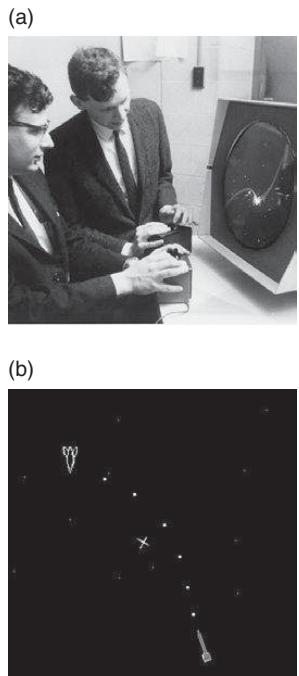


Fig. 9.2. (a) Dan Edwards (left) and Peter Samson, two of the original student developers at MIT, playing Spacewar! on the PDP-1 display in 1962. (b) Spacewar! screenshot.

and most famous of these computer games. It was created in 1962 by Steve Russell ([B.9.1](#)) and Martin Graetz, with other young computer programmers at MIT. The game ran on a PDP-1 minicomputer that had been donated to MIT by Digital Equipment Corporation (DEC) for the students to develop interesting applications – although DEC probably did not expect them to create a video game application. The game was inspired by the “space opera” science fiction of E. E. “Doc” Smith. It is a two-player game, with each player having control of a spaceship and attempting to destroy the other using photon torpedoes. The gravity field of the sun, in the center of the screen, pulls on both ships, and players need to avoid falling into it. In an emergency, a player can enter hyperspace and return to a random location on the screen. Spacewar! was later used by engineers at DEC as a test program on every new PDP machine before shipping it. The DEC sales force also distributed Spacewar! with newly installed DEC computers, and many interesting variants of the program were developed.

From these early beginnings, the development of games on university computers grew rapidly, mainly through students writing experimental game software in their spare time. The *Star Trek* TV series, first shown in 1966, created a loyal fan base and inspired several computer game versions. One of the most popular of these “find and fight Klingons” *Star Trek* games was written by an eighteen-year-old schoolboy named Mike Mayfield. He had access to a Sigma 7 computer at the University of California, Irvine, and he wrote the game in BASIC during the summer of 1971. The game delighted almost everyone who saw it, and it was ported (transferred from one system to another) and modified to run on many different computers. Mayfield produced a version in HP BASIC that Hewlett-Packard put into the public domain and made available on tape. DEC also distributed their version of the code, which became the basis for versions of *Star Trek* that ran on personal computers like the Apple II ([Fig. 9.3](#)), the Commodore, and the BBC Micro.

Don Daglow ([B.9.2](#)), a student at Pomona College in California, developed the first interactive baseball game on a PDP-10 in 1971. A few years later, he wrote Dungeon, one of the first role-playing computer games. Role-playing games focus on character development and problem solving rather than action. Daglow’s game was based on the board version of Dungeons and Dragons. The game used text; *line-of-sight graphics*, in which the player had to have a clear view of an object to act on it; and maps of the dungeons to show in which direction the player should explore. Daglow also continued to develop his baseball game, and a version was released for the Apple II in 1981. It was also the basis for



B.9.1. Steve “Slug” Russell was one of the first computer game developers. In 1961 he wrote the first version of the Spacewar! program as a student at MIT. In 1968 he was working for the Computer Center Corporation – informally known as C-Cubed – in Seattle as their hardware chief. It was there that Paul Allen and Bill Gates developed their deep knowledge of the PDP-10 that served them so well when they came to write the BASIC interpreter for the Altair. It was Russell who first introduced Allen to PDP assembler code.

Fig. 9.3. Screen shot of *Star Trek* on the Apple II.



other commercial versions. Daglow went on to become one of the leaders of the new profession of computer game designers.

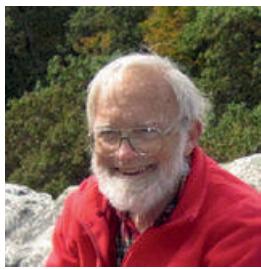
In the early 1970s, William Crowther (B.9.3) worked at Bolt, Beranek and Newman (BBN), a defense contractor that was developing the ARPANET, the precursor to today's Internet. In his spare time, he created a text-based exploration game in FORTRAN on BBN's PDP-10:

I had been involved in a non-computer role-playing game called Dungeons and Dragons at the time, and also I had been actively exploring in caves – Mammoth Cave in Kentucky in particular.... I decided I would fool around and write a program that was a re-creation in fantasy of my caving, and also would be a game for the kids, and perhaps some aspects of the Dungeons and Dragons that I had been playing. My idea was that it would be a computer game that would not be intimidating to non-computer people, and that was one of the reasons why I made it so that the player directs the game with natural language input, instead of more standardized commands. My kids thought it was a lot of fun.²

He called his game Colossal Cave Adventure, or just Adventure. The player explores a virtual cave system by entering simple two-word commands and then reading the new text that the command generates. Crowther released the game on the ARPANET in 1975, and it rapidly became popular among the ARPANET community. Crowther's Adventure was perhaps the first *interactive fiction* game, in which the player helps guide the action. Crowther was contacted a year later by Don Woods (B.9.4), a graduate student at Stanford University, who asked for permission to make enhancements to the game. Adventure was mainly an exploration game, and Woods added many more magical objects as well as creatures like the elves and trolls of author J. R. R. Tolkien's Middle Earth trilogy. This extension of Crowther's original Adventure game is probably the first example of a "mod," short for "modification." Many games produced for PCs are now designed so that technically able users can make modifications and thus add extra interest to the games. A team of students at MIT – Tim Anderson, Marc Blank, Bruce Daniels, and Dave Lebling – were also inspired by Adventure and created Zork, another early interactive fiction game. "Zork" was MIT hacker slang for an unfinished program. Although the game was originally

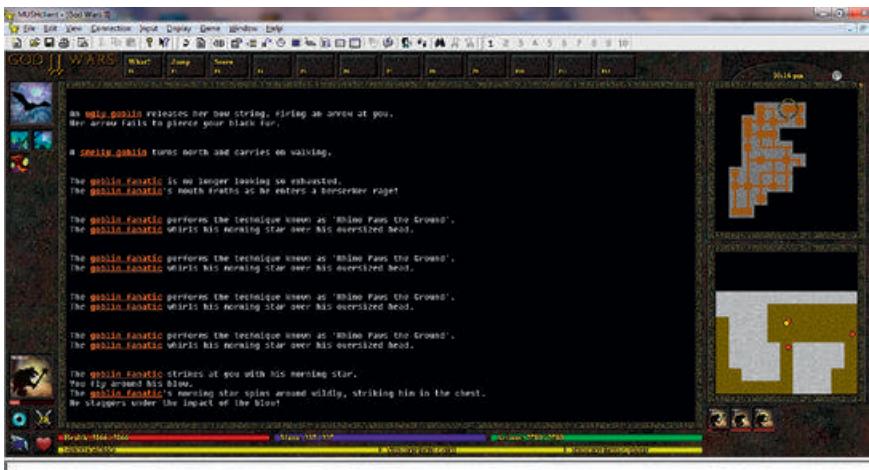


B.9.2. Don Daglow is an eminent game designer who has been associated with many landmark computer games, such as *Star Trek*, *Dungeon*, *Baseball*, and *Utopia*.



B.9.3. William Crowther was part of the BBN team that built the ARPANET. He was an ardent caver and wrote the Colossal Cave Adventure computer game to entertain his daughters.

Fig. 9.4. Dungeon crawling in a typical MUD game.



written for the PDP-10 minicomputer, the authors recognized the potential of text games for personal computers and founded Infocom in 1979. Several commercial versions of Zork were released.

Roy Trubshaw, a student at the University of Essex in England, was in turn inspired by Zork to create a multiuser version of an adventure game for his university's PDP-10 computer. He called his game MUD (Fig. 9.4), for "Multi-User Dungeon," and made the game available to the outside world through a guest account accessible using the Internet. MUD was the first multiplayer online role-playing game and spurred the development of many similar games. The name MUD became used for a variety of multiplayer adventure games on the Internet. In a typical game, players read descriptions of rooms, objects, and other players, and interact with them by typing commands such as "hit the troll with the Elvish sword." The object of the game is to explore a fantasy world, slaying monsters and completing quests along the way, and gaining skills and special powers.

In the early 1970s, these games were mainly text based, using words rather than graphics. Players used teletypes as computer terminals to play these games on a university mainframe or minicomputer. In the mid-1970s, teletypes began to be replaced by computer monitors with much faster output capabilities and a much more flexible graphical user interface. With the advent of personal computers, computer games became a main driver for innovation in computer-generated graphics.



B.9.4. Don Woods was a co-author of the Colossal Cave Adventure game. As a student at Stanford he discovered an early version of the Adventure. He was immediately hooked by the game and decided to extend it by adding new features.

Arcade video games

Russell's Spacewar! was the inspiration for a coin-operated version called Galaxy Game that was installed in a student union building on the Stanford University campus in 1971. That same year, Nolan Bushnell (B.9.5) and Ted Dabney, founders of the Atari computer game company, produced another variant of Spacewar! called Computer Space. This was the first computer arcade video game, equipped with a coin-slot mechanism, but it was not a great



Fig. 9.5. Pong was Atari's first computer video arcade game.

commercial success. The game that was a success, and which led to Bushnell and Dabney setting up Atari, was Pong, a simulated table tennis game where each player controls the position of a paddle.

The first example of a computer table tennis game had been created by physicist William Higinbotham in 1958 and was called Tennis for Two. The game display was an oscilloscope screen that had been programmed to show a side view of a tennis court. The game was played by two players each using controllers to “hit” the ball over the net. The players had to take into account the effect of gravity on the tennis ball when deciding on the trajectory of each stroke. The game was used to entertain visitors to Brookhaven National Laboratory on Long Island, New York. Bushnell had the idea to produce a similar tennis game as an arcade game. With engineer Al Alcorn, Bushnell built a prototype, added a coin box, and installed it in a bar in Sunnyvale, California, to test customer reactions to the game. It was a stunning success: people started coming into the bar, not to buy beer, but to play the game. The game was played by two players using a “paddle” to hit the ball as it rebounds from a wall. Atari shipped the first version of Pong in November 1972 and by 1973 had more than \$3 million in sales (Fig. 9.5). Many imitators of Pong soon appeared, and by 1977 there was a glut of Pong clones on the market.

After the success of Pong, Bushnell and his colleague Steve Bristow had the idea to create a single-player game in which the player would try to hit a ball against a wall and destroy the wall, brick by brick. Steve Jobs was commissioned to design a prototype of what became the game Breakout. For a fee of \$750, Jobs promised to deliver a working version within four days. He enlisted the help of Steve Wozniak and, after a marathon effort with little or no sleep, Woz delivered the Breakout prototype on time.

It was a Japanese game designer, Tomohiro Nishikado (B.9.6), who created the first video game to depict human-to-human combat and thus began the debate about violence in video games. In Japan, the game was called Western Gun and was released by Taito Corporation in 1975. The game could be played in single-player or two-player versions and was the first video game to show an actual gun on the screen. It also introduced dual joystick controls, levers that could be moved to control the action on the screen, one for the movement of the player and the other for the shooting direction. The Japanese version of the arcade game used special-purpose hardware, but the U.S. version was rewritten for the Intel 8080 microprocessor. It was released by Midway Manufacturing Company in the United States under the title Gun Fight. This was the first microprocessor-based arcade video game, and its success in the United States opened the way for Japanese games to enter the American market.

In 1977, Nishikado decided to use a microprocessor to design and implement a new arcade game called Space Invaders. This turned out to be one of the most successful shoot ‘em up games, in which the player shoots at targets, in this case multiple waves of aliens descending from the top of the screen. Space Invaders was a huge commercial success and is said to have caused a coin shortage in Japan. More than 360,000 arcade units were sold worldwide, and by 1982 the game had earned more than \$2 billion in quarters and 100-yen coins. The game is said to have begun the “golden age of arcade video games.”³



B.9.5. Nolan Bushnell, founder of Atari – the name is a reference to the game of Go where the call *atari* is similar to *check* in chess. Bushnell had first become interested in computers as an engineering undergraduate at the University of Utah when he took a course on computer graphics given by the graphics pioneers David Evans and Ivan Sutherland.



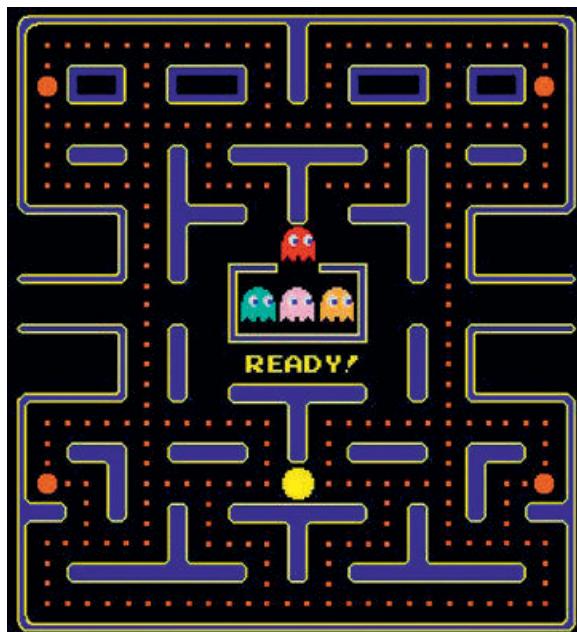
B.9.6. Tomohiro Nishikado is one of the most well-known Japanese video game developers. His first games were the early team sports games, *Soccer* and *Davis Cup*, both released in 1973. Nishikado claims that *Soccer* was Japan's first video game and that his 1974 driving race game, *Speed Race*, was "possibly the first Japanese game in America." His most successful game, *Space Invaders*, was released in 1978 and began "the golden age of video arcade games."^{B1}

The success of Space Invaders was followed by similar games such as Asteroids from Atari and Galaxian from Namco Limited. Color was introduced into arcade games at the end of the 1970s. In 1980, Namco succeeded in creating a new genre of video game in addition to shoot 'em up and sports games. The game was Pac-Man (Fig. 9.6), a maze game in which the player controls a character, Pac-Man, through the twists and turns of a maze, eating various objects on the way. When all the objects have been eaten, the player proceeds to a new stage after an intermission in which Pac-Man is chased by four different enemies. The game's designer, Toru Iwatani (B.9.7), said that he had designed the enemies with distinct personalities and behavior to keep the game from being boring but also to ensure that it was not impossibly difficult to play. The game generated a new audience for video games because it appealed to female players as well as the traditionally male gaming community. The game sold more than 350,000 units and had an estimated thirty million active players in 1982. A perfect Pac-Man game occurs when a player eats all available objects and vanquishes every enemy on all of 255 levels. The first person to achieve this was Billy Mitchell of Hollywood, Florida, in 1999, who finished the game in about six hours. Arcade games continued to be popular throughout the 1980s but began to decline in popularity as home game consoles and personal computers became more popular in the 1990s.

Console wars

In 1951, the U.S. inventor Ralph Baer was developing television technologies for an electronics company based in New York. Instead of just using patterns on a television screen to calibrate the equipment, he realized that it was possible to give the viewer the ability to manipulate what was displayed on the

Fig. 9.6. Pac-Man is one of the most influential video games of all time. It was the first game to introduce a character as a gaming icon, and it established a new genre of games featuring mazes. It was the first game to include "power-ups" – in which the characters get extra powers or abilities for a short time. It also introduced the idea of "cut scenes" – brief animations or videos that give more detail about the character and the game. Pac-Man is credited with opening up video gaming to female audiences.





B.9.7. Toru Iwatani was born in Tokyo and joined the computer software company Namco in 1977. He led the team that designed and built the arcade version of what was called Pac-Man in the United States, and the game was first released in Japan in 1980. The game quickly became an international success and was one of the first video games to appeal to players of both sexes.

screen. Although the company was not interested in pursuing his idea, Baer did not give up. In 1966, now working for a different company, he and colleague Bill Harrison developed a game called Chase. This used a “console” – a special-purpose box of electronic circuitry – to allow the user to control images displayed on a TV screen. This was the first video game to use a standard television for its display. The technology was licensed to a company called Magnavox, which released the world’s first home video game console, the Magnavox Odyssey, in 1972. Through a system of plug-in devices called *cartridges*, the same console was able to play a small number of different games, including table tennis and various shooter games. After a TV advertising campaign that starred Frank Sinatra, Magnavox sold more than one hundred thousand consoles in 1972. During its entire life span, more than two million Odyssey systems were sold.

In these early consoles, the games were hard wired into the electronic circuitry, as in the early arcade machines, and it was difficult to add new games. Bushnell was frustrated by this limitation and wanted to produce a flexible video game console that could play all of Atari’s current games. However, to bring such a console to market, Bushnell needed more funding so he sold Atari to Warner Communications in 1976. However, after a series of disagreements with Warner managers, Bushnell left the company in 1978. By the second half of the 1970s, the software for video games was being developed for consoles that contained microprocessors. Game cartridges were now programs burned into ROM (read-only memory) chips that could be plugged into a slot on the game console. In 1977, the Atari 2600 was released offering nine different games and soon became one of the most popular consoles of the time. However, it was not until Atari made a console version of Space Invaders that the company had the first “killer app” for video game consoles. Atari released its last console, the Atari Jaguar, in 1993. It was not a commercial success, and sales of the console ended in 1996.

In 1985, the Japanese company Nintendo reenergized the video game console market with the release of its Nintendo Entertainment System (NES). The hardware came bundled with two controllers and the game Super Mario Bros. This game was designed by the Japanese game designer Shigeru Miyamoto (B.9.8). He based Mario on a character, originally called Jumpman, that he had used in his earlier arcade game success Donkey Kong. After the failure of Nintendo’s Radar Scope arcade game in the United States in 1980, the runaway success of Donkey Kong had saved the company from financial collapse. For the game Super Mario Bros. on the new NES console, Miyamoto gave Mario a big mustache and Italian nationality, and based him in New York City because of its “labyrinthine subterranean network of sewage pipes.”⁴ In the game, Mario has to explore eight different worlds and defeat many enemies to rescue Princess Peach. Jumping to access places is a key ability in Mario games. The game also used the idea of “power-ups.” Mario can acquire three different power-ups: the Super Mushroom, which causes him to grow larger; the Fire Flower, which allows him to throw fireballs; and the Starman, which gives him temporary invincibility.

Nintendo’s Super Mario Bros. game was the first successful “side-scrolling” game. Side scrolling is a computer graphics technique in which the action is viewed from the side and the on-screen character moves from left to right through the scene, but can also go backward as well as forward. This technique is typically used for what are called “platform” games – action games featuring characters



B.9.8. Shigeru Miyamoto is the creator of the Donkey Kong, Super Mario Bros., and The Legend of Zelda franchises for Nintendo. He is one of the most successful game designers and is often referred to as “the father of modern gaming.”⁸²

that run, jump, and climb over obstacles through the different levels of the game. Donkey Kong, Miyamoto’s earlier game with Mario as a character, was the first game that allowed players to jump over obstacles and cross gaps. Since his appearance in 1985, the character Mario has appeared in more than one hundred different games and is the anchor for what is now the best-selling video game franchise of all time with more than two hundred million games sold.

Miyamoto followed up his success with another innovative game for the NES console, The Legend of Zelda, released in 1986. In this game, Miyamoto deliberately decided to focus more on puzzles and riddles than just adventure and battle scenarios. Exploration was also a key feature of the game. He took inspiration from his childhood in Kyoto, Japan:

When I was a child, I went hiking and found a lake. It was quite a surprise for me to stumble upon it. When I traveled round the country without a map, trying to find my way, stumbling on amazing things as I went, I realized how it felt to go on an adventure like this.⁵

Miyamoto had heard of Zelda Fitzgerald, wife of the famous novelist F. Scott Fitzgerald, and decided to name Princess Zelda after her because he thought the name sounded “pleasant and significant.” The Legend of Zelda is the fourth best-selling title for the NES console. It was the first console game that allowed players to stop playing and save the state of the game for them to resume later.

In 1991, the Sega Corporation introduced Sonic the Hedgehog for their Genesis console. This was a platform game like Mario, and the blue hedgehog Sonic soon became the mascot of Sega’s video game business. The Sonic the Hedgehog franchise made Sega’s console very competitive in the early 1990s – at one point they had 65 percent of the market in North America. However, they faced strong competition, first from Nintendo, and then from Sony, when the PlayStation console was launched in 1994. The PlayStation or PS1 was the first console to sell more than one hundred million units. Faced with such competition, and with the Microsoft Xbox launching in 2001, Sega dropped out of the console market in 2001. The company has since focused on producing video games, including their Sonic the Hedgehog franchise, for other console manufacturers.

By the early 2000s, there were only three major game console suppliers left in the market – Nintendo, Sony, and Microsoft. Nintendo had introduced new consoles at regular intervals after the NES with the Super Nintendo Entertainment System (SNES) in 1991, Nintendo 64 in 1996, and the GameCube in 2001. Although the GameCube was profitable, its worldwide sales of twenty-two million units placed it well below Sony in popularity. Sony had introduced the PlayStation 2 (PS2) console in 2000, and within a few years the console had sold more than 150 million units, making it the best-selling console of all time. The video game series Grand Theft Auto, which put players in the role of gangsters, had been a major success for Sony’s PS1 console, and the company secured a brief exclusive on the PS2 for Grand Theft Auto III, the groundbreaking three-dimensional version of the game.

Microsoft released its Xbox console with a game called Halo: Combat Evolved, or simply Halo (Fig. 9.7). Halo was a first-person shooter (FPS) game, in which the player aims and shoots at targets seen from the viewpoint of the main character. Halo was extremely successful in North America and Europe, and Microsoft



Fig. 9.7. Halo: Combat Evolved was the launch title for the Microsoft Xbox in 2001. The initial game has spawned many sequels as well as a prequel in Halo: Reach. The writer Brian Bendis has compared the cultural effect of Halo to that of the Star Wars movies.

followed this by launching its Xbox Live service in 2002. This service uses the Internet (see [Chapter 10](#)) to support online gaming among multiple players. Halo 2 was released in November 2004 and soon became the most popular online game. By June 2006, more than five hundred million games of Halo 2 had been played with more than seven hundred million hours played on Xbox Live. However, although Xbox sold more than twenty-four million units, comparable to the sales of Nintendo's GameCube, both were well behind the sales of Sony's PS2.

Competition in the manufacture of video game consoles continues to be fierce. Sony introduced the PlayStation 3 (PS3) in 2006, but its unconventional hardware architecture with IBM's novel cell processor made it a difficult platform for game developers. The PlayStation 4, launched in 2013, returned to a more conventional microprocessor chip based on the Intel x86 architecture. The Microsoft Xbox 360, introduced in 2005, competed directly with the PS3, and by 2013 both consoles had sold more than seventy million units. Nintendo launched the innovative Wii in November 2006. The Wii incorporates a handheld pointing device that can detect motion in three dimensions. This enabled Nintendo to broaden the appeal of their console beyond the core gaming community and the Wii sold more than eighty million units, outselling the Xbox 360 and the PS3. With the launch of Kinect in 2010, the Microsoft Xbox gaming console introduced another mode of interaction by allowing gamers to control the Xbox using spoken commands and gestures. After Kinect had sold eight million units in the first sixty days after launch, *Guinness World Records* recognized it as the “fastest selling consumer device.”⁶ The competition between the Microsoft Xbox One and the PS4, both launched in 2013, promises to be interesting to watch.

Computer graphics and video games

The term *computer graphics* was first used in 1960 by William Fetter, a graphic designer at the Boeing Company. Much of the early research on computer graphics was inspired by the Semi-Automatic Ground Environment (SAGE) air defense project at MIT's Lincoln Laboratory. The SAGE system used a CRT monitor for display and a light pen device developed at the Lincoln Lab for user interactions with the screen. In 1959, Wesley Clark and others at the laboratory created the TX-2 computer. Ivan Sutherland, then a graduate student at MIT, developed Sketchpad, a revolutionary graphics software program that allowed users to draw simple shapes on the TX-2 computer screen using a light pen. In 1967, David Evans recruited Sutherland to the computer graphics research group at the University of Utah. During the 1970s, many of the important breakthroughs in computer graphics research originated from the Utah research group.

Computer displays are built up from two-dimensional grids of small rectangular cells called *pixels* – “picture elements.” The picture is built up from these cells, and the smaller and closer the pixels are together, the better the quality – or the “higher the resolution” – of the image. Modern displays and printers are *raster devices*, which produce an image by scanning it as a series of lines. A *raster scan* is the pattern of parallel lines that construct an image on a CRT screen. A raster graphics image or *bitmap* is just the rectangular grid of pixels that make up the image. The bitmap corresponds bit-for-bit with the image displayed on the screen and is characterized by the width and height of

the image in pixels and by the number of bits per pixel. For color images, each pixel needs to hold a minimum of three numbers – to specify the intensity of the three red, green, and blue components. Raster graphics images cannot be scaled up in size because the image will appear *pixelated* – that is, the individual rectangular pixels making up the image will become visible.

An alternative way of representing images in a computer program is to use *vector graphics*. This technique uses simple geometrical shapes such as points, lines, curves, and rectangles to represent images. Thus a square would be represented by just four points, one for each corner. Each of these points has information that tells the computer how to connect the points – with straight lines in the case of a square – and what color to use to fill in the enclosed shape. Vector graphics images can be resized with no loss of detail – the vector points are just spread out or shrunk as required and the computer can easily redraw the image. However, for printing, vector graphics images need to be converted to a bitmap/raster format. The SAGE air defense system was one of the first to use vector graphics displays.

In the early 1980s, the computer graphics technology used by game designers for the 8-bit microprocessor personal computers of the time was quite primitive compared to the state of the art in computer graphics research. Nowadays, the hardware technology in game platforms has advanced so much that computer game companies are now some of the leaders in graphics research. We illustrate some of the early techniques used by game developers for personal computers by looking at how games were implemented on the Atari 800 and the Commodore 64.

One of the standard devices for platform video games such as Mario was side scrolling, in which the characters move across a background screen. Many of these 8-bit microprocessor-based computers offered hardware support for scrolling and for *sprites*, small graphic elements of fixed width and height that can be positioned independently on the main screen (see Fig. 9.8). Atari's 2600 video game console had hardware support for up to five sprites that could be moved independently of the game background. The Atari 800 home computer had a similar hardware capability and supported four sprites eight pixels wide to represent characters and one sprite eight pixels wide that could optionally be split into four two-pixel wide sprites to represent missiles. The characters could be moved horizontally and vertically, and the software specified priorities – that is, which image would be visible – whenever two sprites ran into each other. The computer hardware also supported scrolling the background by up to fifteen pixels in the horizontal and vertical directions. Larger scrolling movements required shifting the start of the screen display in memory. The Commodore 64 had similar graphics capabilities and provided hardware support for scrolling and for eight sprites. Both the Atari and the Commodore supported a palette of sixteen colors, and the Atari also provided eight *luminance* settings. Luminance specifies the amount of light emitted from a given area and is an indicator of how bright the surface will appear.

By the 1990s, as predicted by Moore's law, microprocessors had become cheaper and more powerful, and the two-dimensional tricks of these early video games gave way to true three-dimensional models. Instead of two-dimensional shapes like circles and rectangles, three-dimensional models allow a *wireframe*

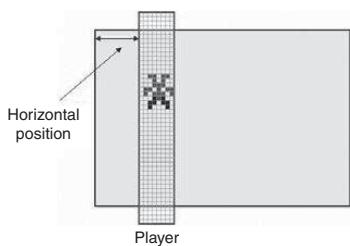


Fig. 9.8. Sprites were graphics devices used to represent characters or missiles in early computer games. This figure shows the Player/Missile sprite for the Atari 800 home computer.

representation of any object. A wireframe model uses points, lines, and curves to show the edges of an object, including the opposite sides and any internal features that would be normally hidden from view. A whole range of techniques is now available to generate all sorts of complex shapes and curves. The file specifying the game scene contains the geometry of the objects in the scene and their relative positions. The game designer also has the freedom to view the scene from different perspectives and to specify different choices of lighting. The process of converting the resultant three-dimensional model to a two-dimensional image on a screen is called *rendering*. The computer has to calculate which surfaces are behind any given object from the viewer's perspective and which should therefore be hidden when the two-dimensional image is created. The rendering process also adds lighting and surface texture effects. The graphics research group of Evans and Sutherland in Utah created the first *hidden-surface algorithms* to realistically display overlapping objects – that is, sets of rules to determine which surfaces would not be visible from a certain viewpoint and so should be hidden.

Computer graphics is now the foundation for whole new industries. Computer animation is the process of using computers to create moving images, and computer-generated imagery (CGI) is now a standard technology for the movie industry. In 1995, Pixar Animation Studios released *Toy Story*, the first full-length computer-generated animation movie. Computer-aided design (CAD) uses computer graphics to assist the design process in many industries, including automotive, aerospace, and shipbuilding. Computer graphics technologies also underpin modern scientific and information visualization.

The modern era of computer games

In 1991, Sid Meier's (B.9.9) Civilization game was released for the PC. This is a single- or multiplayer strategy game in which players attempt to build an empire in competition from other civilizations and under attack from marauding barbarians. Each player starts with one settler unit and one warrior unit and by exploration, warfare, and diplomacy, attempts to become the dominant civilization. The game begins in 4000 B.C. before the Bronze Age and can continue to 2050 with a future space-age civilization expanding to settle new stars. As time advances during the game, players can choose to invest in new technologies – from the wheel and pottery in the early stages, to nuclear power and spaceflight near the end of the game. Wise investments in science and technology can often bring decisive advantages for a civilization, as in real life. In 1996, *Computer Gaming World* magazine chose Civilization as the best game of all time, explaining:

While some games might be equally addictive, none have sustained quite the level of rich, satisfying gameplay quite like Sid Meier's magnum opus. The blend of exploration, economics, conquest and diplomacy is augmented by the quintessential research and development model, as you struggle to erect the Pyramids, discover gunpowder, and launch a colonization spacecraft to Alpha Centauri.... Just when you think the game might bog down, you discover a new land, a new technology, another tough foe – and you tell yourself, "just one more turn," even as the first rays of the new sun creep into your room ... the most acute case of game-lock we've ever felt.⁷



B.9.9. Sid Meier on stage at the Game Developer Conference in 2010. Meier was born in Ontario, Canada, and graduated from the University of Michigan. He cofounded the game companies MicroProse and Firaxis Games. His hugely successful game, Sid Meier's Civilization, was released by MicroProse in 1991.

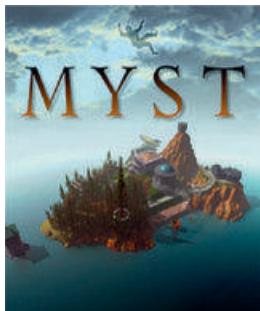


Fig. 9.9. *Myst* is an adventure game during which the player explores the virtual world of a mysterious island. It introduced a new type of puzzle-based adventure game.

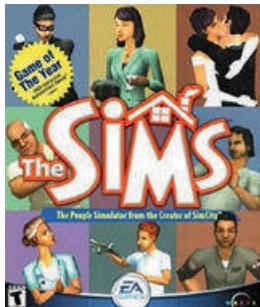


Fig. 9.10. *The Sims* is a life simulation game. It has become one of the most popular computer game franchises worldwide. The franchise has sold over 175 million copies (August 2013).

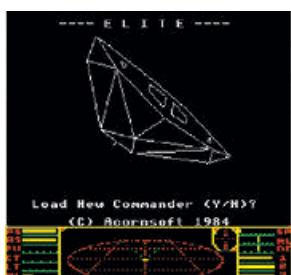


Fig. 9.11. *Elite* was written for the BBC Micro by two Cambridge University undergraduates. It was the first game to introduce three-dimensional graphics.

The real-time strategy (RTS) genre of computer games came of age in 1992 when Westwood Studios released *Dune II*. An RTS game allows the player to issue commands to vast armies as the action unfolds. The game continues to run even if the player is not actively giving commands. The plot of *Dune II* is loosely based on the 1965 science fiction novel *Dune* by Frank Herbert and on the 1984 movie of the same name. The goal of the game is to harvest valuable “spice” from the deserts of the planet Arrakis at the same time as fighting off enemy troops and avoiding the giant sandworms that can destroy the huge spice harvester. From the profits made from trading spice, players can generate more units and build up military bases. The game starts with an overhead view of a map with the “fog of war” covering all areas not within sight of the player’s units. As the units explore, the fog lifts and more of the map is revealed. The game features a model economy for the trading of resources and the building of bases as well as detailed management of military units. *Dune II* became the standard template for future RTS games and was the inspiration for Westwood Studios’ later *Command & Conquer* video game franchise.

The game *Myst* (Fig. 9.9) introduced a new style of puzzle-based adventure games and became a surprise hit. It was developed by the brothers Robyn and Rand Miller in Spokane, Washington, and was released for Apple’s Macintosh computer in 1993. *Myst* went on to become the best-selling PC game in the 1990s, and its sales remained the highest until overtaken by the life-simulation game *The Sims* in 2002 (Fig. 9.10). *Myst* was one of the “killer apps” that made CD-ROM drives a standard feature on PCs. (CD-ROM stands for “compact disk read-only memory.”) CD-ROMs are made from plastic coated with a thin layer of aluminum to make a reflective surface. A CD stores binary data as a series of microscopic “pits” separated by “lands.” The changing intensity of the pattern of reflected light from a laser shone onto the disk can be converted back to binary data. CDs can store up to 700 megabytes (MBs) of data, a huge increase in storage capacity compared to a floppy disk with a capacity of only a few MB. (MB stands for megabyte or 1,000,000 bytes of storage.) Further improvements of this technology have led to DVDs with a capacity of five or more GB. (GB stands for gigabyte or 1,000,000,000 bytes.) DVDs are now used for distributing copies of movies. The movie industry has also introduced Blu-ray Discs, which have a capacity of 25 GB and allow the distribution of high-resolution versions.

In 1984, two Cambridge University undergraduates, David Braben and Ian Bell, created a game called *Elite* (Fig. 9.11) for the BBC Micro. The BBC Micro was built by Acorn Computers Ltd. for a British Broadcasting Corporation project to promote computer literacy in the United Kingdom. *Elite* combined a traditional combat game with space trading between star systems. While making a hyperspace jump between stars, the trading ship could be attacked by enemy ships. Players could gain credits by trading, asteroid mining, piracy, and bounty hunting. They could use these credits to upgrade their ships with better weapons or increased cargo capacity. The major technical innovation introduced in *Elite* was its use of three-dimensional wireframe graphics. This technique created a genuine three-dimensional world and allowed realistic movement in all directions. A galaxy in *Elite* contained 256 planets to explore, and the game’s universe consisted of eight galaxies. The player could follow the state of the game on a three-dimensional radar display. For the program to fit into the 14



Fig. 9.12. A visual representation of the World of Warcraft.

kilobytes of memory available on the BBC Micro, Braben and Bell had to write it in the low-level microprocessor assembly language. The success of the game with its revolutionary three-dimensional graphics meant that it was ported to all popular computers as well as to Nintendo's NES console.

During the 1990s, the increasing power of microprocessors and the falling price of computer memory made three-dimensional graphics much easier to implement. The 1993 game Doom from id Software was one of the first to take advantage of this technology advance (B.9.10). The game not only introduced more realistic three-dimensional graphics but set the ground rules for FPS video games. In the game, the player is a space marine who has to fight his way through hordes of invading demons from hell. Doom's graphic violence and its satanic imagery generated considerable controversy. The game was played by more than ten million people within two years of its release.

In 1996, a company called 3dfx Interactive introduced Voodoo Graphics, an affordable three-dimensional *graphics accelerator card* for personal computers. Graphics accelerator cards have a specialized processor that performs three-dimensional rendering, which otherwise would take up large amounts of computing power, thus freeing up the main CPU for other tasks. A modern *graphical processing unit* (GPU), a computer chip that performs rapid mathematical calculations for drawing images, now plays the same role of accelerating the graphical rendering process. This increase in graphical processing capability allowed Roy Trubshaw's original concept for text-based MUDs to be transformed to graphical multiplayer games. With the increasing availability of the Internet, these graphical MUDs evolved into a genre known as *massively multiplayer online role-playing games* (MMORPGs). The first game in the Warcraft (Fig. 9.12) series, Warcraft: Orcs and Humans, was released in 1994. A decade later World of Warcraft was published, and by 2009 World of Warcraft had more than ten million subscribers. RuneScape, a free MMORPG, was developed by Andrew and Paul Gower in Cambridge, England, and was released in 2001. As of November 2010, RuneScape was credited by *Guinness World Records* as having more than 175 million registered users.

Minecraft is a recent example of a game that has “gone viral” – become a huge success – with no publisher backing and without any commercial advertising (Fig. 9.13). The original PC version of the game was developed by Swedish programmer Markus “Notch” Persson (B.9.11). Minecraft released an “alpha” version in 2009 and the full release in November 2011. In less than one month, more than one million copies of the game had been sold. By March 2012,



B.9.10. John D. Carmack (left) and John Romero are two of the founders of id Software, a Texas-based video game development company. Carmack and Romero were responsible for introducing new realism into PC games with their three-dimensional graphics technology. Their breakthrough game was Wolfenstein 3D in 1992, followed by Doom in 1993 and Quake in 1996. Doom defined the FPS genre and was the first PC game to be ported to Linux.

Fig. 9.13. A screen shot of Minecraft. The game runs on multiple platforms such as PCs, consoles, and smart phones. Markus Persson designed the game just for fun, and never expected that it would achieve such popularity. Initially the game just featured a landscape with a collection of blocks and few people would have predicted its runaway success. Majong, Persson's company that develops Minecraft has no publicity department and the news about the game is spread by word of mouth - or through the Internet. The game is a *sandbox* game and gives the players freedom to modify the world that they are playing in.



Minecraft had become the sixth best-selling PC game of all time. As of February 2014, the game has sold more than fourteen million copies for the PC and more than thirty-five million copies on all platforms. In 2012, Minecraft was the most purchased title on Xbox Live Arcade and the fourth most played title on Xbox Live. Minecraft - Pocket Edition was developed for the Android platform and has now reached more than twenty-one million copies.

What is the reason for the phenomenal success of Minecraft? Part of the reason for its success is its seemingly endless variety and extensibility. At its most basic, it is just a survival/construction game - a sort of digital Lego - in which the players can build various three-dimensional structures from textured cubes. They can also grow crops, raise farm animals, and collect the resources necessary for survival. There are mines where players can dig shafts and search for iron ore, coal, gold, and gems. The iron ore can be smelted to craft a pickaxe, sword, or armor. These tools come in handy for fighting the zombies, spiders, dragons, and the other hostile creatures that pervade the landscape. In multiplayer mode, players can participate as a team and join together to construct buildings or entire cities, share resources, or fight enemies. Another element of Minecraft's success is the ability to incorporate user-generated content and a wide variety of mods are available for download from the Internet. Minecraft has turned out to be not only "just another creativity game" but has also been used in a number of educational applications, including history and science. The game has also been used to sketch out ideas for architectural and urban design. In 2012, Cody Sumter, then a research assistant at the MIT Media Lab, said "Notch hasn't just built a game. He's tricked 40 million people into learning a CAD program."⁸ Google is even using the game to entice bright students to learn about quantum mechanics:



B.9.11. Markus "Notch" Persson is the Swedish creator of Minecraft and cofounder of Mojang, the game company he formed in 2010. He started programming at the age of seven on a Commodore 128 personal computer and developed a text-based adventure game when he was eight. Notch was his login ID when he started playing computer games and the nickname stuck. Despite the phenomenal success of Minecraft he still considers himself as a "garage developer" who writes games for fun and not necessarily for profit.



Fig. 9.14. Nintendo's Game Boy handheld video game device was introduced in 1989.

Millions of kids are spending a whole lot of hours in Minecraft, not just digging caves and fighting monsters, but building assembly lines, space shuttles, and programmable computers, all in the name of experimentation and discovery. So how do we get these smart, creative kids excited about quantum physics? We talked to our friends at MinecraftEdu and Caltech's Institute for Quantum Information and Matter and came up with a fun idea: a Minecraft modpack called qCraft. It lets players experiment with quantum behaviors inside Minecraft's world, with new blocks that exhibit quantum entanglement, superposition, and observer dependency.⁹

The 1980s saw the beginnings of the video games industry with much experimentation and many new games being developed. During the 1990s, the games industry matured and the graphical images and animations became increasingly realistic. Games began to be designed by large teams rather than individuals, with a corresponding increase in costs. Because of the increased focus on visually richer and more complex games, there has been a decline in the number and variety of new games being produced. This trend has continued into the twenty-first century with an increasing emphasis on a small number of very successful game franchises to reduce the risk of expensive failures. Minecraft is an encouraging exception to this trend. And technology still has some surprises: the inexorable progress predicted by Moore's law and the rapid growth of smart phones has created new opportunities and challenges for the computer games industry.

Angry Birds and casual gaming

The availability of more powerful microprocessors together with advances in screen technologies allowed the development of handheld gaming devices not requiring connection to a TV. Nintendo led the way with the introduction of the Game Boy (Fig. 9.14) in 1989. One of the games that came bundled with the Game Boy was the puzzle game Tetris (Fig. 9.15), in which users have to manipulate falling blocks in a variety of shapes to form complete lines. The game is recognized by *Guinness World Records* as being "the most ported game in the history of video games,"¹⁰ and versions of Tetris have now appeared on more than sixty-five different gaming platforms.

Mobile phones became a gaming platform with the introduction of the game Snake by Nokia in 1997. Snake was originally developed in the 1970s. The player controls a snake that moves around the screen with the objective of avoiding hitting its own tail. As the snake moves around, it eats food and its tail grows longer, making it increasingly difficult to avoid. After Nokia preloaded the game onto its phones, the popularity of Snake exploded. This was one of the first examples of a "casual game" – a game with limited complexity that can be played while waiting or traveling. Because of the small phone screen and the very limited amount of memory and computing power available on mobile phones, these early casual games needed to be very simple.

The cost of computing technology has now become low enough to make smart phones and tablets with as much computing power as a PC widely available. Casual games explicitly designed for short play sessions are now among

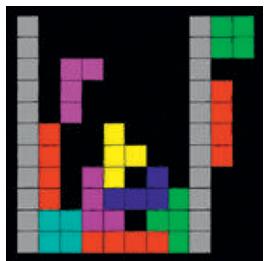


Fig. 9.15. Tetris was designed and programmed by Alexey Pajitnov while he was working at the Soviet Academy of Sciences' Computing Center in Moscow, Russia, in 1984.



Fig. 9.16. The idea for Angry Birds came from one of Rovio's designers, Jaako Iisalo. He created a screen shot showing a cartoon flock of round birds looking cross. One of the directors of Rovio, Niklas Hed said: "People saw this picture and it was just magical."¹¹

the most popular applications for smart phones. One of the most popular of these games is Angry Birds (Fig. 9.16), developed by the Finnish company Rovio Entertainment Limited. In the game, players launch cartoon birds from slingshots to destroy cartoon pigs. The two cousins who run the company, Mikael and Niklas Hed, realized in early 2009 that the smart phone market for games was about to take off. The cousins had developed more than fifty games before Angry Birds and, armed with this experience, they had set out to create a game that was entertaining for ordinary people, not just for hard-core gamers. When Niklas saw his mother distracted from cooking the Christmas dinner by playing Angry Birds, he realized they had a hit on their hands. He said, "She doesn't play any games. I realized: this is it."¹¹ The game became available in Apple's App Store in December 2009, and more than twelve million copies had been sold by 2013.

The casual game FarmVille introduced a new social element into mobile gaming by its association with the Facebook social networking site. The game was launched by Zynga in June 2009 and reached more than ten million daily users within six weeks. The game was inspired by a Chinese farming simulation game called Happy Farm that allowed players to grow crops, trade with others, sell produce, and steal from neighbors. At its peak, the game had more than twenty million daily users in China and Taiwan. Although Happy Farm and FarmVille have now declined in popularity, many new games are now available. In 2011, in the United States and Western Europe there were more than sixty million casual gamers on Apple's iPod touch, iPhone, and iPad platforms who downloaded an average of 2.5 games a month. Half of all paid and free downloads on Apple's App Store in these countries were games. On Facebook, more than 50 percent of users play games and there are more than 250 million people playing games on the site every month.

Worryingly, nearly 20 percent of Facebook users say they are addicted to playing games. Video game addiction is now becoming a real problem in many countries. During the last decade, there have been a handful of horrific deaths caused by serious video game addiction – deaths of players and of children being neglected while their parents played games. In 2005, a young man in South Korea died after playing a video game continuously for fifty hours. In 2010 a woman in the United States was convicted of second-degree murder after she told investigators that she had shaken her baby to death because its crying had interrupted her playing FarmVille. As early as 1981, a bill called the Control of Space Invaders (and other Electronic Games) Bill had been introduced into the British Parliament and was only narrowly defeated. In 2007, the Chinese government introduced restrictions on online gaming that required Internet games operating in China to identify users by their resident identity numbers. After three hours of continuous play, players under eighteen are prompted to stop playing the game and take some physical exercise. If they continue the game, their game points are reduced and, after five hours, all their game points are erased. This legalistic approach is just one attempt to address the serious problem of addiction that is accompanying the continuing rise in popularity of computer video games.

Keyconcepts

- Genres of computer games
- Action games

- Ball and paddle
- Space opera
- Maze
- Platform
- Adventure games
- Simulation and management
- Real-time strategy or RTS
- First-person shooter or FPS
- Role-playing games or RPGs
- Multiuser dungeons or MUDs
 - Massively multiplayer online role-playing games or MMORPGs
 - Casual games
- CD-ROMs, DVDs, and Blu-ray disks
- Computer graphics technologies
 - Pixels and bitmaps
 - Raster graphics
 - Vector graphics
 - Side-scrolling and sprites
 - 3-D modeling
 - Rendering



Stewart Brand and Rolling Stone

In 1972, ten years after Slug Russell created Spacewar!, Stewart Brand, a writer and computer enthusiast who had worked with Doug Engelbart on the “Mother of All Demos” in 1968, watched students at the Stanford Artificial Intelligence Laboratory enthusiastically playing the game. The game ran on a PDP-10 computer costing \$500,000 and differed greatly from IBM-style batch computing. Although the game was not yet personal computing, it was clearly a very personal use of the PDP-10’s time-sharing system. Students were using the computer for fun, interactively, with no thought for the cost of the computer time, just as personal computers would be used only a few years later. Brand investigated the personal computing and gaming culture further, and Bob Taylor allowed him to talk to the researchers at Xerox PARC. The result was that PARC researchers and their laid-back, freewheeling style of research appeared in a story in *Rolling Stone* in December 1972 (Fig. 9.17). The story was titled “SPACEWAR: Fanatic Life and Symbolic Death Among the Computer Bums” and featured photographs by Annie Liebowitz and an interview with Alan Kay. Needless to say, this publication caused considerable embarrassment back at Xerox’s conservative East Coast headquarters and resulted in the company ordering a more formal system of computer access at the Palo Alto laboratory.



Fig. 9.17. Stewart Brand's “Spacewar” article in *Rolling Stone*.

The photograph shows the Teletype Corporation's Model-33 terminal also known as an ASR-33 (ASR stands for Automatic Send Receive). The model was introduced in 1963 and more than half a million teletypes had been produced by 1975. The 500000th machine was plated with gold and exhibited.

Teletypes

The teletype (Fig. 9.18) was an electromechanical typewriter developed in the last century that could be used to send and receive typed messages using point-to-point connections. Teletypes were quickly adapted to provide a text-based user interface to the early mainframe computers and minicomputers. Although they were soon replaced by punched card readers and fast line printers for most purposes, teletypes continued to be used as interactive time-sharing terminals. It was not until computer terminals with monitors – video display screens – became widely available in the mid- to late 1970s that the teletype finally became obsolete. However, some of their heritage still lives on. When video displays became available, these could display thirty lines of text in a few seconds instead of the minute or so required for printing on paper. On a teletype, users typed commands after a *prompt character* was printed. Initially, games kept the same user interface for the video display screen and this is why the *command line interface* and *prompts* used by professional software developers look the way they do today.



Fig. 9.18. The photograph shows the Teletype Corporation's Model-33 terminal also known as an ASR-33 (ASR stands for Automatic Send Receive). The model was introduced in 1963 and more than half a million teletypes had been produced by 1975. The 500000th machine was plated with gold and exhibited.

10 Licklider's Intergalactic Computer Network

It seems reasonable to envision, for a time 10 or 15 years hence, a “thinking center” that will incorporate the functions of present-day libraries together with anticipated advances in information storage and retrieval.... The picture readily enlarges itself into a network of such centers, connected to one another by wide-band communication lines and to individual users by leased-wire services. In such a system, the speed of the computers would be balanced, and the cost of gigantic memories and the sophisticated programs would be divided by the number of users.

J. C. R. Licklider¹

The network is the computer

Today, with the Internet and World Wide Web, it seems very obvious that computers become much more powerful in all sorts of ways if they are connected together. In the 1970s this result was not so obvious. This chapter is about how the Internet of today came about. As we can see from Licklider's (B.10.1) quotation beginning this chapter, in addition to arguing for the importance of interactive computing in his 1960 paper on “Man-Computer Symbiosis,” Lick also envisaged linking computers together, a practice we now call *computer networking*. Larry Roberts, Bob Taylor's hand-picked successor at the Department of Defense's Advanced Research Projects Agency (ARPA), was the person responsible for funding and overseeing the construction of the ARPANET, the first North American *wide area network* (WAN). A WAN links together computers over a large geographic area, such as a state or country, enabling the linked computers to share resources and exchange information. As Roberts said later:

Lick had this concept of the intergalactic network which he believed was everybody could use computers anywhere and get at data anywhere in the world. He didn't envision the number of computers we have today by



B.10.1. Joseph C. R. Licklider (1915–90) was a visionary computer pioneer whose impact is still felt everywhere in computer science. His interests were wide ranging and included psychological aspects of communications and learning, brain studies, computer networks, time-sharing computers, interactive systems, and cooperation between computers and humans. His groundbreaking paper “Man-Computer Symbiosis” investigated the possibility of a closer cooperation between humans and computers with computers being used to augment human intellectual capacity. His 1968 paper with Bob Taylor, “The Computer as a Communications Device,” outlined their joint vision of what has become the present-day Internet.



B.10.2. Sun Microsystems was founded in the early 1980s by two Stanford MBAs, Vinod Khosla and Scott McNealy, and a Stanford graduate student, Andy Bechtolsheim, together with another graduate student, Bill Joy, from the University of California, Berkeley. Sun was an acronym for Stanford University Network, where Bechtolsheim's prototypes were already running, connected by the Ethernet.

any means, but he had the same concept – all of the stuff linked together throughout the world, that you can use a remote computer, get data from a remote computer, or use lots of computers in your job. The vision was really Lick's originally. None of us can really claim to have seen that before him nor [can] anybody in the world. Lick saw this vision in the early sixties. He didn't have a clue how to build it. He didn't have any idea how to make this happen. But he knew it was important, so he sat down with me and really convinced me that it was important and convinced me to move into making it happen.²

It was Roberts and a small team of dedicated engineers and graduate students, mostly trained at MIT, who built the ARPANET. The ARPANET was rapidly followed by a worldwide proliferation of similar – but incompatible – networks. In 1974, Bob Kahn and Vint Cerf brought order to this chaos by publishing a paper with the intimidating title “A Protocol for Packet Network Intercommunication.” Their paper coined the term *Internet* for what they called the “internetworking of networks.”

In 1982, two Stanford MBAs, Vinod Khosla and Scott McNealy, founded Sun Microsystems, together with hardware expert Andy Bechtolsheim, a PhD student in Stanford University’s Electrical Engineering Department, and a Unix software expert, Bill Joy, a graduate student from the nearby University of California, Berkeley (B.10.2). They set up the company to develop robust single-user workstations, computers with less computing power than minicomputers but more powerful than PCs. From the start, the Sun founders envisioned networking their workstations. As Sun’s CEO, Scott McNealy, said,

The whole concept of “The Network is the Computer” we started at Sun was based on the fact that every computer should be hooked to every other computing device on the planet.³



Fig. 10.1. An advanced version of a telegraph with a printing receiver and transmitter.

To begin our story, we shall go back in time to the origins of the telegraph (Fig. 10.1) system and briefly describe what has been called “The Victorian Internet” by author and journalist Tom Standage.

Fig. 10.2. A map of the global communications network of the British Empire in 1872. Telegraph cables reached as far as Hong Kong and Australia.



The Victorian Internet

The preface to Standage's book includes a summary of the impact of the new technology:

During Queen Victoria's reign, a new communications technology was developed that allowed people to communicate almost instantly across great distances, in effect shrinking the world faster and further than ever before. A worldwide communications network whose cables spanned continents and oceans, it revolutionized business practice, gave rise to new forms of crime, and inundated its users with a deluge of information. Romance blossomed over the wires. Secret codes were devised by some users and cracked by others. The benefits of the network were relentlessly hyped by advocates and dismissed by its skeptics. Governments and regulators tried and failed to control the new medium. Attitudes towards everything from news to diplomacy had to be completely re-thought. Meanwhile, out on the wires, a technological subculture with its own customs and vocabulary was establishing itself.⁴

Although this sounds like a description of the present-day Internet, it is in fact referring to the global telegraph network, which transformed business and personal life in the nineteenth century, over a hundred years earlier than the Internet (Fig. 10.2). Just as the Internet of today is sometimes known as the "Information Superhighway," the nineteenth-century telegraph network was called the "Highway of Thought."

Message routing systems, in which a message is passed from one "station" to the next, have operated since antiquity. The mechanism for passing the messages has changed, however. At first, human runners carried the messages. In 1791, the French engineer Claude Chappe (B.10.3) invented a sophisticated optical signaling system that consisted of a network of towers. An operator in each tower moved two large, jointed arms on a signaling device called a *semaphore* to spell out messages. Each tower also had two telescopes, one pointing backward and the other forward. Because Chappe's semaphore network was an optical system, it required good line-of-sight visibility between the towers. A major drawback of the system was that it only worked well in good weather,



B.10.3. French postage stamp with Claude Chappe (1763–1805), inventor of the optical telegraph.

Fig. 10.4. The route of the famous Pony Express in the United States. This mail service showed that fast transcontinental communications were possible, but the service was expensive and was overtaken by telegraph technology along the same route. The Pony Express operated for less than two years, from April 1860 to October 1861.

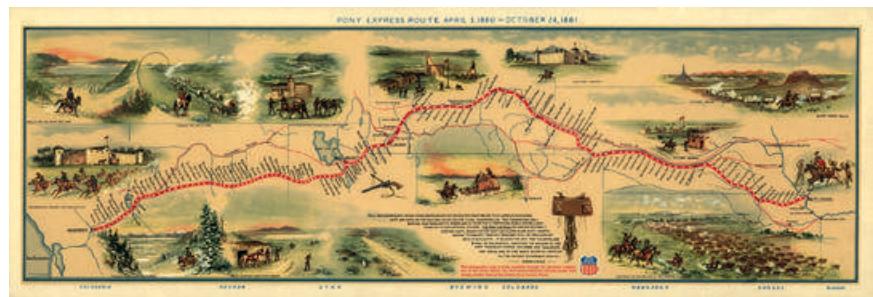


Fig. 10.3. One of Claude Chappe's telegraph towers was rebuilt near Saarbrucken, in Germany. The French engineer Chappe succeeded in covering France with a network of 556 stations stretching a total distance of 4,800 kilometers. The network was used for military and national communications from 1792 to the 1850s.

and then only during the daytime. When Napoleon Bonaparte came to power in 1799, he quickly recognized the military value of such a rapid communication system and arranged for a portable version of Chappe's semaphore signaling equipment to be developed. In searching for a name for his invention, Chappe devised the word *telegraphe* from two Greek words meaning *to write at a distance*. His semaphore system operated with a countrywide network of towers in France for more than fifty years (Fig. 10.3). Finland, Denmark, Sweden, Russia, and the United Kingdom quickly established similar optical telegraph networks. At the height of the network's popularity, nearly a thousand semaphore towers operated across Europe.

For all the success of Chappe's system, it was superseded by a superior technology after little more than fifty years: telegraph technology based on using electric signals that traveled over cables. Before the telegraph network became fully operational, couriers on horseback used to deliver long-distance messages. In the United States, the Pony Express linking Missouri with California operated for only slightly more than a year between 1860 and 1861, but could not compete for long with the telegraph (Fig. 10.4). The electric telegraph had much greater speed and reliability, and operated in all weather conditions. In the United Kingdom, it was an entrepreneur named William Fothergill Cooke who, in an uneasy collaboration with physics professor Charles Wheatstone, worked hardest to raise enthusiasm for building an electric telegraph network to span the country (B.10.4). At about the same time, in the United States, Samuel Morse (B.10.5), a painter and scientist, was also working tirelessly toward the same objective. The electric telegraph sent messages using a code of dots and dashes that we now call Morse code. Although the specific telegraph technology used in the United Kingdom and the United States differed in detail,



B.10.4. Entrepreneur William Cooke (1806–79) and physicist Charles Wheatstone (1802–75) pioneered the telegraph in the United Kingdom. After a slow start, undersea cables were laid connecting the whole of the British Empire.



[Fig. 10.5.](#) Memorial in Telegraph Field, Valentia Island, Ireland, commemorating the site of the European end of the first transatlantic cable in 1858.



[B.10.5.](#) Photograph of the statue of Samuel F. B. Morse (1791–1872), pioneer of the telegraph and inventor of Morse code, in New York's Central Park. The statue was dedicated in 1871.

the telegraph network expanded rapidly in both countries. By 1850, there were more than two thousand miles of wire in the United Kingdom and more than twelve thousand miles in the United States. In 1852, the first underwater cable linking London and Paris was laid across the English Channel. In the United States, the idea of a transatlantic cable had been suggested by Morse in the 1840s but the idea was thought to be impractical. In 1854, the wealthy businessman Cyrus Field ([B.10.6](#)) took up the idea and had a cable laid from New York to St. John's in Newfoundland, preparing to extend the cable across the Atlantic Ocean to Ireland ([Fig. 10.5](#)). Field persuaded both the U.S. and British governments to back the project, and the first undersea connection was established in August 1858 ([Fig. 10.6](#)). Queen Victoria sent the first transatlantic message to U.S. President James Buchanan.

Unfortunately, the engineer in charge of the project, Edward Whitehouse, had very little understanding of the science of underwater telegraphy. Within a month, the cable had failed. A joint government inquiry was set up to determine the reasons for the failure. It was William Thomson, then a physics professor at Glasgow University, later to become Lord Kelvin, who put submarine telegraphy on a sound scientific foundation. Thomson's understanding of the relevant physics was confirmed by the successful laying of an undersea cable through the Persian Gulf in 1864, connecting Europe to India. One of the key technologies for undersea cables was the use of a rubbery gum called *gutta-percha*, obtained from a tree grown in Southeast Asia, to coat and protect the cables ([Fig. 10.7](#)). The London-based Gutta Percha Company suddenly found itself with a virtual monopoly on the production of submarine cables. The Gutta Percha Company eventually became part of Cable & Wireless Worldwide.

The telegraph system was an early example of a *store-and-forward* network, in which messages were sent to an intermediate station before they were transmitted to their destination. Telegraph services routed signals through intermediate relay stations because electrical losses from telegraph wires accumulated and degraded a message if it had to travel over long distances. Intermediate stations received messages as dots and dashes of Morse code and recorded them on punched paper tape. Incoming messages were separated from each other by tearing the tape in the appropriate places, which led to the intermediate stations being called “torn-tape relay centers.” A telegraph operator at the receiving station read the destination on the message tape



[B.10.6.](#) Cyrus Field (1819–92) began work at age fifteen as an office boy for A. T. Stewart & Co., New York City's first department store. By the age of twenty, he was a partner in a paper manufacturing company, and at thirty-three was able to retire as a wealthy man. In 1854, Field became enthusiastic about the possibility of laying an undersea transatlantic telegraph cable from Newfoundland to Ireland. After several failed attempts, in August 1858 Field arranged for Queen Victoria to send the first transatlantic message to President James Buchanan, and there were great celebrations on both sides of the Atlantic. The cable and the project were not successfully completed until 1866. In the portrait, Field is seen touching a globe and holding a length of cable wire.

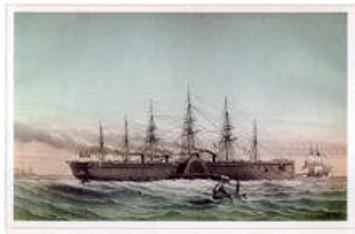


Fig. 10.6. Isambard Kingdom Brunel's Great Eastern was the largest ship afloat in 1865 and was ideally suited to cable laying. On 13 July 1866, the ship left Valentia Bay in Ireland and arrived in Newfoundland two weeks later.



Fig. 10.7. The gutta-percha tree grows in Southeast Asia. It produces a rubbery gum that coped better with the underwater environment than traditional rubber and was used to coat the under-sea cables.



Fig. 10.8. The launch of the Sputnik satellite by the Soviet Union on 4 October 1957 was a wake-up call for the United States. The first Sputnik weighed only 184 pounds and was the size of a basketball. Sputnik II, launched just a month later, weighed half a ton.

and took it to the appropriate transmitter for sending to the next relay center, until the message reached its final destination. At the height of the telegraph's popularity, a major relay station had dozens of inbound receiving and outbound transmitting terminals, with scores of operators and thousands of messages queued up during peak hours. Our present-day computer networks also use a store-and-forward system, not for whole messages but for standard-size pieces of information called *packets*.

Nuclear war and packet switching

The idea of a digital communication network using *packet switching*, which breaks up the message into small packets for transmission, occurred almost simultaneously to two researchers on different continents – but for very different reasons. Paul Baran was a researcher at the RAND Corporation in Santa Monica (B.10.7). The RAND Corporation – RAND is an abbreviation of Research AND Development – was originally part of the Douglas Aircraft Company and was set up as an independent nonprofit research organization in 1948. Baran had previously worked for the Eckert-Mauchly Computer Corporation on the UNIVAC computer and for the Hughes Aircraft Company in Los Angeles on the computerized Semi-Automatic Ground Environment (SAGE) early warning system. Hughes was bidding on a contract for the control system for the Minuteman missiles, and Baran became alarmed, because, as he said later, "You had all these missiles that could go off by anyone's stupidity. The technology was never to be trusted."⁵

When Baran joined RAND in 1959, he set out to study the problem of whether U.S. communication systems could survive a nuclear attack by the Russians. The late 1950s were times of great political tension between the United States and the Soviet Union. Two years earlier, the Soviets had managed to put a satellite into Earth orbit, an event that the United States interpreted as a clear threat to national security (Fig. 10.8). Both the United States and the Soviet Union had large stockpiles of nuclear weapons. For these nuclear arsenals to work as a deterrent to war, it had to be evident to each country that the other country would be able to launch a retaliatory attack after a first strike. This was the doctrine of *mutual assured destruction*, usually known by the acronym MAD. If the early warning systems failed and missiles had exploded on U.S. territory, the president needed to have the "minimal essential communications" to launch a counterattack.

Until Baran's arrival at RAND, there had been little progress on how to protect the U.S. communications network in the event of a nuclear attack. Because of his familiarity with digital computers, Baran found that he thought about the problem very differently from most of his colleagues at RAND. He said, "Many of the things I thought possible would tend to sound like utter nonsense, or impractical, depending on the generosity of spirit in those brought up in an earlier world."⁶ Signals on a telephone network were *analog* signals, in which the amplitude of the signal varied continuously, and the quality of the signal deteriorates as the number of "hops" through a network increased. Baran realized that by using digitized messages generated by a computer, a signal could be stored, replicated exactly, and retransmitted an unlimited number of times. This technique would allow the signal to be transmitted over long distances through intermediate routing stations without any distortion or loss.

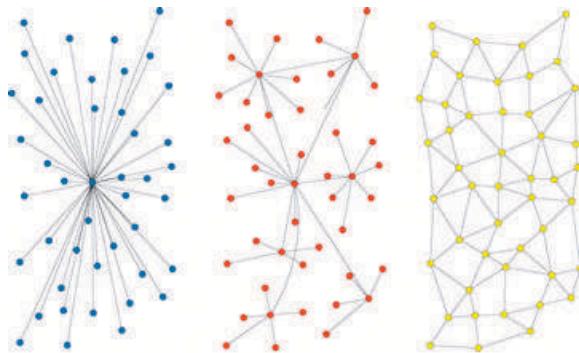


Fig. 10.9. A figure from Paul Baran's original paper on packet switching shows examples of centralized, decentralized, and distributed networks. The distributed network is also called a "fishnet" network, and its most important feature is the lack of centralization so that all nodes have an equal importance. This fishnet architecture also contains a certain level of redundancy because each node is connected to several other nodes. This redundancy allows multiple paths for the message packets to reach their destination. This provides the network with much more robustness and reliability than a centralized network with a single path to the destination node.

In the early 1950s, Warren McCulloch, a neurophysiologist at MIT, had been talking with John von Neumann and others about the brain and its network of *neurons* (nerve cells in the brain). Together with the mathematician Walter Pitts, McCulloch developed a mathematical model of a neuron and linked these model neurons together to form an artificial "neural network" (see Chapter 13). This network was a computer program that operated much like the connected nerve cells in the nervous system. Baran knew of this work and was influenced by McCulloch's ideas of neural networks:

Warren McCulloch in particular inspired me. He described how he could excise a part of the brain, and the function in that part would move over to another part.... McCulloch's version of the brain had the characteristics I felt would be important in designing a really reliable communication system.⁷

At the time, the telephone system depended on a hierarchical system in which switching centers ranked one above the other. Baran had analyzed what would happen to the telephone network after a nuclear attack on U.S. military targets, and his results showed that the system was very vulnerable to even the incidental, "collateral" damage caused by such an attack. In designing a more robust system, Baran introduced what he called "redundancy" – that is, exceeding what is necessary – into the network, by allowing multiple pathways between the different centers. He advocated building a distributed network that looked rather like a fishnet (Fig. 10.9). A network with only one link connecting each center has a redundancy level of one and is obviously extremely vulnerable to disruption. Baran's calculations showed that "just a redundancy level of maybe three or four would permit almost as robust a network as the theoretical limit."⁸ This meant that even if some links in the network were destroyed in a nuclear attack, because each center was connected to three or four other centers it would still be possible to find working paths through the network.



B.10.7. Paul Baran (1926–2011) pioneered the concept of distributed networks and packet switching for reliable communications after a nuclear strike. He was never able to persuade AT&T engineers to build even a prototype of his packet-switching system.

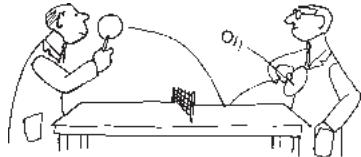


Fig. 10.10. "Interactive Communication consists of short spurts of dialog."⁸

Baran's second idea was just as radical: to introduce a type of redundancy in sending messages. He proposed splitting each message into a number of fixed-length "message blocks" – what we now call *packets* – and allowing the blocks to take independent paths through the network. In an interview with the writer Stewart Brand, Baran later described what he called the "wonderful" properties of this method, called *packet switching*:

Anyway, you send the packet out, and when this station gets it off to the next guy, it sends back, "OK, got it. You can erase the previous one." If the first station doesn't hear back, it sends another copy out in a different direction. The packets can arrive out of order. We just sort them out at the end. Since it didn't have to be synchronous, you didn't have to lock everything all together. It didn't take very long before we started seeing all sorts of wonderful properties in this model. The network would learn where everybody was. You could chop up the network and within half a second of real-world time it would be routing traffic again. Then we had the realization that if there's an overload in one place, traffic will move around it. So it's a lot more efficient than conventional communications. If somebody tries to hog the network, the traffic routes away from them. Packet switching had all these wonderful properties that weren't invented – they were discovered.⁹

In spite of all these advantages, Baran had a hard job trying to convince AT&T – which then operated virtually the entire U.S. telephone network – of the virtues of distributed networks and packet switching. At the time, all telephone networks used what is known as *circuit switching*. When you called someone, the network established a physical connection between the two endpoints and the intermediate links were dedicated to your conversation as long as you stayed on the line. For a phone call, this method makes sense, because there is typically a steady exchange of information during a conversation. By contrast, data communications sent from a user sitting at a computer tend to be "bursty," with large amounts of data sent all at once, followed by pauses with no data being transmitted (Fig. 10.10). Keeping a line reserved for this type of communication is not a very efficient way to use the available bandwidth in the network. *Bandwidth* is the technical term used to define a network's maximum capacity for transmitting information. Using packets that can be routed in multiple ways through a distributed network turns out to be a much more efficient use of the available bandwidth because packets from different messages can use the same links. Because the different packets of the same message can travel over multiple different paths, the packets making up any given message may arrive out of sequence and need to be reassembled in the right order at the receiving center. Each packet must therefore contain a *header* at the beginning of the packet that describes where it is going and to which message it belongs.

In response to criticisms of his ideas from his colleagues, Baran wrote a series of papers that showed in detail how all these problems could be solved. One new factor in understanding his idea was simply the much higher speed of switching that was possible in computer networks. As we have seen, the electromagnetic relay switches of the earliest computers had been replaced by electronic switches capable of much-faster switching rates. To AT&T engineers not familiar with the new digital computer technology, Baran's idea of breaking up

a telephone conversation into packets of information seemed truly ridiculous. He recalled:

The story I tell is of the time I went over to AT&T headquarters – one of many, many times – and there’s a group of old graybeards. I start describing how this works. One stops me and says, “Wait a minute, son. Are you trying to tell us that you open the switch up in the middle of the conversation?” I say, “Yes.” His eyeballs roll as he looks at his associates and shakes his head. We just weren’t on the same wavelength.¹⁰

By 1965, Baran had managed to persuade the U.S. Air Force to back the creation of a trial distributed switching network. Unfortunately, AT&T declined any involvement in the project and the proposal never got off the ground. Baran reluctantly moved on to study other problems.

At about the same time that Baran was abandoning his research on packet switching, Donald Davies at the United Kingdom’s National Physical Laboratory (NPL) was also thinking about packet switching and computer networks (B.10.8). His interest stemmed from reasons very different from Baran’s concerns about network survivability in a nuclear war. Davies had joined NPL in 1947 to work on Alan Turing’s ambitious project to build the Automatic Computing Engine – the ACE computer. Turing became frustrated by the delays and bureaucracy at NPL and left to join the University of Manchester’s computer team, leaving Davies in charge of producing a less ambitious version of ACE. Davies’s team delivered the Pilot ACE machine in 1950, and the English Electric Company later successfully marketed a commercial version called the DEUCE computer. After a visit to MIT in 1954, Davies became interested in the problem of sending data communications over a network. While at MIT, he had seen that a significant problem with time-sharing computers was the cost of keeping a phone connection open for each user. Instead of being concerned about the survivability of networks after a nuclear strike, Davies was thinking about the efficient support of online data processing in which users at computer keyboards were generating the data. In 1965, he wrote in an internal NPL note outlining the problem:

Starting from the assumption that on-line data processing will increase in importance, and that users will be spread out over the country, it is easily seen that data transmission by a switched network such as the telephone network is not matched to the new communication needs that will be created. The user of an on-line service wishes to be free to push keys sporadically, and at any rate he wishes, without occupying and wasting a communication channel.¹¹

Davies also realized that delays in store-and-forward distributed networks could be minimized by using short message blocks he called *packets*. He very deliberately introduced the word *packet* to describe the fixed-length, short blocks of information that made up the message and that traveled separately through the network to their destination. The choice of name made clear that packet switching was fundamentally different from traditional message switching. And it was certainly a much better meme than the phrase “distributed adaptive message block switching” used by Baran!



B.10.8. Donald Watts Davies (1924–2000) graduated from Imperial College in London in mathematics and physics. He then joined the United Kingdom’s NPL, where he worked with Alan Turing on the ACE computer. Independent of Paul Baran’s work in the United States, Davies came to almost the same conclusions as Baran about efficient and reliable communication networks. However Davies’s reasons for developing packet-switching networks were related to efficient use of time-shared computers and not for providing network survivability in the event of a nuclear war.

At the end of 1967, one of Davies's colleagues at NPL, Roger Scantlebury, presented a paper on their packet-switching work at a conference in Gatlinburg, Tennessee. Scantlebury's paper outlined a design for a packet-switching network consisting of relay centers called *nodes* connected by digital links. The nodes handled the transmission of the packets between the nodes, and *interface computers* connected the network of nodes to time-sharing computers and to other subscribers. It was from Scantlebury that Larry Roberts first heard about Baran's work on packet switching (B.10.9). He later said of the revelation he experienced at this conference, "Suddenly I learned how to route packets."¹² When Roberts returned to Washington, he hunted down the RAND reports that Baran had sent to ARPA and found time to meet with him in early 1968. Meanwhile, back in the United Kingdom, Davies and his colleagues set about building a small, packet-switching network at NPL. The Mark I version of the NPL network became operational in 1970, and a Mark II version remained in operation until 1986. Despite the similarity of their ideas, Davies always gave credit to Baran as being the first to publish the idea of packet switching in distributed networks, saying, "The honour for [originating packet switching] must go to Paul Baran."¹³

The ARPANET and the third university

When Licklider went to ARPA, he set out to find and fund the leading computing research centers in the country. From these connections, he established an informal circle of advisers, consisting of about a dozen computer scientists from MIT, Stanford University, and the universities of California at Los Angeles and Berkeley as well as from some computer companies. He lightheartedly called this group his "Intergalactic Computer Network." After he had been at ARPA six months, Lick had seen firsthand the diversity and incompatibility of all the computer hardware and software in the research community. As a result, he wrote a memo to the group that raised the possibility of connecting the different computing systems at each site into a network:

Consider the situation in which several different centers are netted together, each center being highly individualistic and having its own special language and its own special way of doing things. Is it not desirable or even necessary for all the centers to agree upon some language or, at least, upon some conventions for asking such questions as "What language do you speak..." It seems to me to be important ... to develop a capability for integrated network operation.¹⁴

By this memo, Lick had extended the idea of his Intergalactic Computer Network from just a group of people to a network of interoperating hardware and software.

When Bob Taylor (B.10.10) became director of the computing program at ARPA in 1966, he inherited this vision of Lick's. Obtaining the funding for such a network was only the first hurdle for Taylor. He now needed a program manager who could actually build the network, and he knew exactly the person he wanted. Taylor had just funded an experiment to connect the TX-2 computer at MIT's Lincoln Laboratory to the System Development Corporation's



B.10.9. Larry Roberts was a graduate in electrical engineering from MIT and went to work on research projects at MIT's Lincoln Laboratory. At the age of twenty-nine, he became the project leader and chief architect of the ARPANET. After leaving ARPA, he moved to the communications industry and founded the first commercial packet-switching network, called TELENET.



Fig. 10.11. MIT's Lincoln Laboratory was established in 1951 to build the nation's first air defense system. However, its roots date back to the MIT Radiation Laboratory, which was formed out of the Physics Department during World War II to develop radar for the Allied war effort.

Q-32 computer, a military mainframe computer in Santa Monica, California. The person in charge of the project was Larry Roberts, an MIT PhD in electrical engineering who, like many others before him, had migrated up the road to the Lincoln Lab. The report on the networking experiment had concluded that although the connection had been successfully established, its reliability and response time to commands were, as later described by Roberts, "just plain lousy." Roberts, with his deep technical knowledge of the problem, was the ideal candidate for the job but, unfortunately for Taylor, Roberts had no interest in leaving Lincoln Lab (Fig. 10.11) to become what he called "just a bureaucrat." In late 1966, nearly a year after he had received the funding for the project, Taylor persuaded ARPA Director Herzfeld to call the director of Lincoln Lab and tell him about ARPA's problem and, at the same time, point out that more than 50 percent of the lab's research funding came from ARPA. Roberts accepted the job two weeks later.

When Roberts took over the ARPANET project in late 1966, he saw that he had three major technical challenges to solve, as well as a surprising "sociological" problem involving human behavior and social relationships. The first challenge was how to physically connect all the time-sharing computers at the different sites. In the experiment between Lincoln Lab and Santa Monica, Roberts had shown that a direct telephone line connection between the two computers could work. The problem was that ARPA had funded more than a dozen major time-sharing computers, and to establish direct connections between all these computers would require more than sixty-five long-distance telephone lines, a number that would rapidly increase and become very expensive as the number of computer systems increased. His second challenge was that even if he had these long-distance lines, how could they most efficiently be used? In their book *Computer: A History of the Information Machine*, Martin Campbell-Kelly and William Aspray state that: "Experience with commercial time-sharing systems had shown that less than 2% of the communications capacity of a telephone line was productively used because most of a user's time was spent thinking, during which the line was idle."¹⁵ The last technical problem that Roberts needed to solve was how all the different incompatible computer systems would communicate with one another without each site having to write many different software interfaces. Although unknown to Roberts at this time,



B.10.10. Bob Taylor graduated in 1958 from University of Texas with a degree in psychology and mathematics. In 1965, at the age of thirty-four, he became director of the Information Processing Techniques Office at ARPA. In this role he was responsible for creating the program that led to the creation of the ARPANET. After leaving ARPA, he went on to be the founding director of the Computer Science Laboratory at Xerox's new PARC. It was at PARC where many of the pioneering computing technologies that we see around us today were invented, guided by Taylor's leadership. See Chapter 8 for a more detailed discussion.

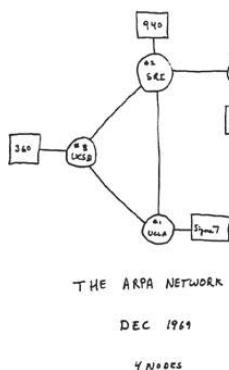


Fig. 10.12. A sketch by Alex McKenzie of the initial four-node configuration of ARPANET connecting the University of California at Santa Barbara, the University of Utah, UCLA, and SRI. The communication backbone of the network was provided by fifty kilobit per second telephone links.

the first two problems had already been solved by Baran and Davies. The use of a store-and-forward network solved the problem of needing to have every computer connected to every other computer. Furthermore, a distributed network, rather than a hierarchical or centralized network, would allow multiple paths between the origin and destination of the messages. Second, the work of Baran and Davies on splitting up the messages into fixed-length packets had shown that a packet-switching network allowed more efficient use of each communication line than a standard message-switching network. Using packets, a single user would not take up the bandwidth of each link, as in a circuit-switched telephone call. Roberts first learned about packet switching and about Baran's work from Roger Scantlebury at the Gatlinburg conference in October 1967.

Roberts had found a solution to his third problem at another conference earlier that year. After a meeting in Ann Arbor, Michigan, in a taxi back to the airport, Wes Clark, another Lincoln Lab veteran, outlined a solution to the problem of each site having to write multiple different versions of interface software for its computer to handle the data communication with the other, different computers. Clark suggested inserting a minicomputer between each mainframe host computer and the network. Each site would then only have to create software to connect its mainframe to the standard network minicomputer. A similar solution had been proposed by the NPL team at the later conference in October. Roberts called the intermediate computers that managed the packet routing and delivery *interface message processors*, or IMPs.

It was at the Ann Arbor meeting that Roberts realized that he also had a sociological challenge on his hands. Many university researchers felt they had worked hard to get funding for "their" computers and were not enthusiastic about having to "waste time" writing software to enable others to use their valuable resources. Roberts noticed a distinct regional bias:

We actually had more conservatism on the East Coast. When I looked for sites that were willing to start, the four West Coast sites were interested and excited to be involved. And the East Coast sites, like MIT, said "Well, I don't want you to touch my computer." So we went with the ones that were cooperative....¹⁶

Two of the groups on the West Coast who were enthusiastic about being connected to a network were Doug Engelbart's team at the Stanford Research Institute (SRI) and Len Kleinrock's Network Measurement Center at the University of California, Los Angeles (UCLA) (B.10.11). Engelbart was working on NLS, his oNLine System for interacting with computers, and he saw the ARPANET proposal as an opportunity to extend his system to support distributed collaboration. Kleinrock had been a PhD student at MIT with Roberts, and in his 1962 thesis had used *queueing theory*, the mathematical modeling of queue lengths and waiting times, to simulate the behavior of store-and-forward message-switching networks. His network simulations had used mathematical models for both the rate of message generation and for the distribution of message lengths. The other two groups who responded favorably to networking their sites were the University of California, Santa Barbara, and the University of Utah, both working on ARPA-funded interactive graphics research. The first four nodes of the ARPANET were therefore all on the West Coast of the United States (Fig. 10.12).



B.10.11. Len Kleinrock ran the Network Measurement Center at UCLA. He is photographed here with the first IMP.

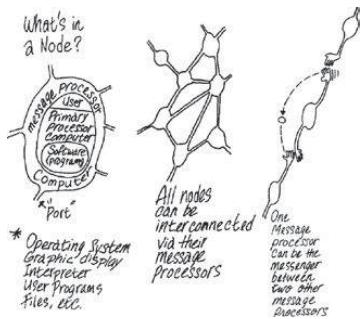


Fig. 10.13. Diagrams of the host nodes and the IMP network: (a) the host nodes; (b) the message processors that allow interconnections through the network; and (c) each message processor can be the messenger between two other message processors.

By early 1968, Roberts had nearly completed a detailed design for the network. There were three basic principles (see Fig. 10.13). The first was that the IMPs should operate as a communication system whose essential task was to transfer packets from anywhere to anywhere else on the network. The IMPs would also take care of route selection and acknowledgment of receipt for the packets. The second requirement was that the network must have a very small message delay time. From his experience with time-sharing systems, Roberts decided that the average transit time through the network must be less than half a second. The third principle was that the IMP system should function independently of whether a host computer had crashed. Network reliability should depend on the IMPs and not the host computers. In addition, the NPL team from the United Kingdom had convinced Roberts that he should specify the use of much faster links than he had specified in his original proposal. By July 1968, Roberts sent out a formal request for quotation to 140 companies giving them information about the project's requirements and inviting them to bid on building the IMPs. One early setback was that two of the major computer companies, IBM and Control Data Corporation (CDC), declined to bid and said that "the network could never be built because there existed no computers small enough to make it cost-effective."¹⁷ Fortunately, Roberts eventually received more than a dozen bids. In December 1968, ARPA announced that it was awarding the contract to build the IMPs to a small consulting firm in Cambridge, Massachusetts, called Bolt, Beranek and Newman, or BBN.

BBN started life as a small consultancy group advising on acoustics in 1948. Richard Bolt and Leo Beranek taught on the MIT faculty, and both were experts on acoustics and, in particular, the acoustics of buildings. Robert Newman, who had been a student of Bolt's and was an architect, joined the consultancy a year later, and BBN was born. The business grew rapidly, and besides its profitable work on buildings, BBN developed unique expertise in the analysis of audiotapes. The company assisted in the analysis of the film of the shooting of President John F. Kennedy and of the shooting deaths at Kent State University. Most famously, in the wake of the Watergate scandal, the White House and the special prosecutor's office called in BBN to examine the 18½-minute gap in the Nixon White House tapes. Dick Bolt headed the investigating committee, which concluded that the erasure was deliberate. By the time that BBN recruited Licklider in 1957, the company had a well-established hiring philosophy summarized by Beranek: "I had the policy that every person we hired had to be better than the previous people."¹⁸ BBN was also well known for its policy of hiring MIT dropouts. The rationale was that if the person had managed to get into MIT, he or she must be smart, and, if the person had subsequently dropped out, all this meant was that he or she could be hired more cheaply than someone who had graduated. Because of the company's recruiting policies and because of the absence of any academic tenure process and any teaching commitments, BBN became a very attractive place for researchers to work. In this way, BBN acquired the informal reputation of being the "third university" in Cambridge, along with Harvard and MIT.

When Licklider joined the company, he convinced Beranek that they should buy him a computer. Beranek later said, "I decided that it was worth the risk to spend \$25,000 on an unknown machine for an unknown purpose."¹⁹ The gamble



B.10.12. Bob Kahn studied at the City College of New York and earned his doctorate at Princeton University in 1964. After working at Bell Labs and then as a professor in electrical engineering at MIT, he took leave from MIT to work at BBN. With Frank Heart, Kahn put together the successful proposal for BBN to build the ARPANET. Kahn is also co-author, along with Vint Cerf, of the TCP/IP protocols that underpin the present-day Internet. The main idea of the protocol is that networks should be connected using gateways that can translate the packets moving between different networks.

paid off and the expertise that the company developed in computing soon became a major asset. BBN later bought the first PDP-1 from Digital Equipment Corporation, the first relatively inexpensive computer that could be operated by a single person. Licklider and his team used the machine to develop one of the first time-sharing systems capable of supporting four simultaneous users. In 1966, BBN went to the Lincoln Lab to recruit an engineer named Frank Heart to work on a hospital computer project. Heart was a graduate from MIT who had taken MIT's first-ever course in computer programming. He had also earned a master's degree while working on the Whirlwind project, the precursor to the SAGE system, and then followed many other MIT graduates in moving to Lincoln Lab to work on its portfolio of exciting real-time computing projects. By the mid-1960s, when BBN recruited Heart, his Lincoln Lab colleagues were the acknowledged experts in building real-time, interactive computing systems.

ARPA's request for quotation for building the IMP network arrived at BBN in the summer of 1968. Bob Kahn, a professor of electrical engineering was at BBN on leave from MIT (B.10.12). Kahn was an applied mathematician who had been working on communications and information theory and wanted some real-world engineering experience. With his work on the ARPANET, Kahn certainly achieved his goal! Kahn had already sent some of his papers to Bob Taylor at ARPA and the request for proposals landed first on his desk. The original hospital project for which Heart been recruited had not materialized, so he and Kahn were tasked with putting together a team and a proposal to bid for the contract. Heart was fortunate to recruit a strong team of experienced engineers, many of whom had worked with him at Lincoln Lab – Will Crowther, to lead the software team; Severo Ornstein, to lead the hardware effort; and Dave Walden, who had four or five years' experience in programming real-time systems. Other key members of the team were Bernie Cosell, whom BBN called “an ace de-bugger whom every BBN manager had learned to rely on if their projects got into trouble,”²⁰ and Ben Barker, an engineer from Harvard who played a vital role in getting the actual hardware for the first IMPs debugged and working (B.10.13).

To simplify the design, Heart had insisted on “a clean boundary between the host responsibilities and the network responsibilities.”²¹ In their book *Where Wizards Stay Up Late*, Katie Hafner and Mathew Lyon summarize the role of the IMP as follows:



B.10.13. A group photo of the IMP team at BBN in 1969. From left to right, Truett Thatch, Bill Bartell (Honeywell), Dave Walden, Jim Geisman, Bob Kahn, Frank Heart, Ben Barker, Marty Thrope (next to Heart), Willy Crowther, and Severo Ornstein. Team member Bernie Cosell is not in the photograph.

Between Roberts and BBN it was settled: The IMP would be built as a messenger, a sophisticated store-and-forward device, nothing more. Its job would be to carry bits, packets, and messages: To disassemble messages, store packets, check for errors, route the packets, and send acknowledgements for packets arriving error-free; and then reassemble incoming packets into messages and send them up to the host machines – all in a common language.²²

From his personal experience, Heart knew the value of building into a system as much reliability as possible. He therefore had the team investigate “error-control mechanisms” to cope with random mistakes in data transmission. Due to electrical noise on the lines, a “1” in a message packet could sometimes change to a “0,” or vice versa. Fortunately, electrical engineers have developed a number of cunning techniques for not only detecting such errors but also, at some additional cost, correcting them. The basic idea is that of a *checksum*, a small number used to detect whether errors have occurred. The checksum is calculated from the bits of the packet at the source and transmitted along with the packet. On arrival, the checksum is recalculated. If the original and recalculated checksums do not agree, a transmission error due to noise has been detected. There are many types of checksums and error-correction techniques now available. The simplest is a *parity check*, a method of detecting errors by counting the evenness or oddness of the number of bits in the packet. This simple parity check requires only a single extra bit of information to be transmitted but can detect only an odd number of errors. More sophisticated methods such as *Hamming codes*, named for the mathematician Richard Hamming, require additional bits. These techniques can not only detect the precise location of the error but also correct it. For the ARPANET, the BBN team decided on a very simple, pragmatic solution: if an IMP detected an error in a packet it would just discard the packet and not send back any acknowledgment of receipt. The source IMP would wait for an acknowledgment and if it had not received one after a certain time, it would resend the packet.

One of Kahn’s responsibilities was to specify exactly how the host computers would interact with the IMP machine. His BBN Report 1822 issued in the spring of 1969 told the host sites how to write a piece of software called a *device driver* to implement the host-IMP interface. At UCLA, Kleinrock had put graduate student Steve Crocker in charge of the programming effort to connect the university’s Sigma-7 mainframe host computer to the IMP, along with fellow graduate students Vint Cerf, Jon Postel, and Charley Kline. Cerf later recalled: “It was a little funny because we were just graduate students. We kept expecting that professional managers would show up and tell us what to do. But they never did, so we just went on our merry way.”²³

Kahn’s report also made clear that the IMP would not contain software for performing host-to-host communication. The responsibility for the host-to-host communications was effectively taken on by an informal collection of graduate students from the first four host sites. They called themselves the Network Working Group and initiated a series of notes called “requests for comments,” or RFCs. Because network users needed to sign a collective agreement to enable each system to work with other systems, the group introduced the word *protocol* into the language of networking. In ancient Greece,

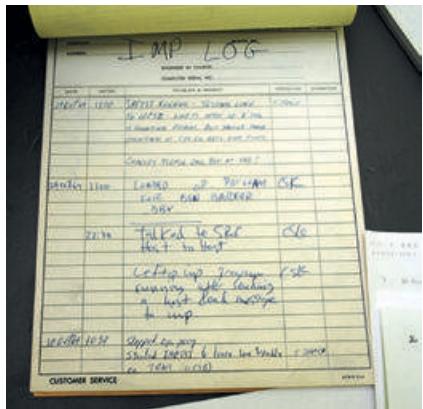


Fig. 10.14. The first message sent over the ARPANET on 29 October 1969. This was an attempt to log into a remote computer at the SRI from UCLA.

a protokollon was the first page of a manuscript that contained a brief summary of the contents, the date, and who authenticated the document. In networking, the meaning of *protocol* resembled the role of the packet header, which specified the destination and information needed to reconstruct the message. A more informal definition was suggested by Cerf: “The other definition of protocol is that it’s a hand-written agreement between parties, typically worked out on the back of a lunch bag, which describes pretty accurately how most of the protocol designs were done.”²⁴

Although the team did not complete a draft of the host-to-host protocol until the summer of 1970, the group had early on adopted a “layered” approach and had also written a couple of key applications. The lowest layer of the protocol specified how to move packets from host to host as a stream of unidentified bits, regardless of what sort of data the bits might represent. The two key applications were one for transferring files and another for *remote login*, software to enable a user at one computer to *log in* – that is, to identify himself or herself at a remote host computer and begin using the site. The final version of the protocol for exchanging files over the network, the File Transfer Protocol, or FTP as it is now usually abbreviated, was not completed until July 1972 when Jon Postel issued the Networking Group’s RFC #354. The remote log-in application was called Telnet and was also very widely used. This application, together with the extension of the IMP interface to allow terminals as well as host computers to send data into the network – an interface called the Terminal IMP controller or TIP – later set the scene for the rapid expansion of the network.

The first IMP node was delivered to Len Kleinrock’s team at UCLA on schedule in September 1969. The software written by Steve Crocker’s team and the hardware interface designed and built by another UCLA graduate student, Mike Wingfield, worked perfectly for the host-to-IMP connection. But it was not until the second IMP was delivered to Engelbart’s team at SRI in October that BBN was able to test intercomputer communication between two different host computers, the Sigma-7 at UCLA and the Scientific Data Systems 940 at SRI (Fig. 10.14). The system worked perfectly and the network was soon extended to the sites at Santa Barbara and Utah (see Fig. 10.12 in the preceding text). From the diagram, it is clear that the only link to Utah was through SRI, meaning that this first prototype distributed network was not yet what Baran had called “a robust web of redundant interconnections.” By 1973, the ARPANET had expanded to become a much more redundant network (Fig. 10.15).

One of the concerns that Bob Kahn had about BBN’s design of the IMP network software was about the flow control of the packets across the network. He said, “I could see things that to me were obvious flaws. The most obvious one was that the network could deadlock.”²⁵ Deadlock is a situation in which the system gets into a state where no action is possible. The scenario that most worried Kahn was one caused by congestion at a destination IMP. If the storage buffers, the areas of computer memory used to temporarily store the information while it was being moved from one place to another, became too full at the receiving node, the packets containing the instructions to reassemble the messages would not be received. The destination IMP would be full up with packets unable to be assembled into complete messages. The pragmatic engineer

Fig. 10.15. The ARPANET topology in September 1973 including links to London and Hawaii.



Crowther and the theoretician Kahn had debated the problem, and pragmatism had won out. Now that the first four nodes were up and working, Heart gave permission for Kahn to fly out to UCLA with Dave Walden to test his theories. Kahn's first experiment demonstrated that the problem was real within only a few minutes of sending out specific patterns of packets. He later said, "I think we did it in the first twelve packets. The whole thing came to a grinding halt."²⁶ As a result of Kahn's results, BBN had to redesign the control system so that enough space was always reserved in the IMP memory buffers for reassembly of incoming packets. Under the redesign, the sending IMP would check that there was sufficient space and, if necessary, delay sending the next message.

Email: The ARPANET's killer app

Bob Taylor had proposed the ARPANET to provide interactive access to ARPA-funded computers across the United States and save money for ARPA by sharing resources. Larry Roberts believes that both of Taylor's goals were achieved:

By 1973, I had cut our computer budget to 30 percent of what it would have been if I hadn't had the network. And saved more money than the network cost. Because I could share computers all across the world and not have to buy computers for every research group that wanted one.²⁷

Although it is clear that the ARPANET did enable some sharing of resources, such cooperation was not its primary use. As Licklider and Taylor had pointed out in their paper on "The Computer as a Communication Device," one of the primary uses of a network would be for communication. A 1973 ARPA report showed that three-quarters of all traffic on the network was for what it called "E-mail," or more commonly nowadays called *email*. Time-sharing computers had had email systems linking users of the same computer for some time. BBN engineer Ray Tomlinson realized that he could extend this idea to send emails between different computers using the ARPANET ([B.10.14](#)):

Once we had the ability to transfer a file from one machine to the other, it became fairly clear that one thing you could do was just write the file across the network and send mail to somebody else. I also happened to be working



B.10.14. Ray Tomlinson was an engineer at BBN who had developed some early email systems for time-sharing computers. When he thought about extending this idea to messages across the ARPANET, he needed a way to separate the name of the recipient in the email address from the machine the recipient was using. He chose the @ sign.

on a piece of software to be used to compose and send mail, called “send message.” And it seemed like an interesting hack to tie those two together to use the file-transfer program to send the mail to the other machine. So that’s what I did. I spent not a whole lot of time, maybe two or three weeks, putting that together and it worked.²⁸

Note Tomlinson’s uncritical use of the word *hack*: in these early days it was a term of respect for some technically clever programming exploit. To be called a “hacker” was a compliment and did not have the unfavorable connotations the word now has. Once email between different sites became possible, Kahn said, “It had tremendous benefits: overcoming the obstacles of time zones, messaging multiple recipients, transferring materials with messages, simple collegial and friendly contacts.”²⁹ Use of email grew rapidly and completely transformed the nature of collaboration. In an early experiment, an email sent to 130 people all around the United States at 5 P.M. generated seven responses within ninety minutes and twenty-eight responses in twenty-four hours. Such a response time now seems very slow, but in the 1970s it was revolutionary.

Tomlinson also got to choose a symbol for designating email addresses: his choice has become an icon for the networked world. Tomlinson needed a symbol to separate the name of the user from the machine that the user was using. He said: “The one that was most obvious was the ‘@’ sign, because this person was @ this other computer, or, in some sense, he was @ it. He was in the same room with it anyway. And so it seemed fairly obvious and I just chose it.”³⁰

Although email usage took off like wildfire, there was much debate about the need to establish a separate email transmission protocol that was independent of FTP. In 1975, the first electronic discussion group called MsgGroup was established. The group had many heated online debates about email headers and requests for comments, and the practice of email *flaming*, sending an angry, critical, or abusive email, became an occasional feature of this and other discussion groups.



B.10.15. Bob Metcalfe and David Boggs invented and built a LAN technology that they called Ethernet to link up the PARC Altos and Gary Starkweather's laser printer. Metcalfe later left PARC to found 3Com, a computer networking company, with the three Cs standing for computers, communication, and compatibility.

From Hawaii to the Ethernet

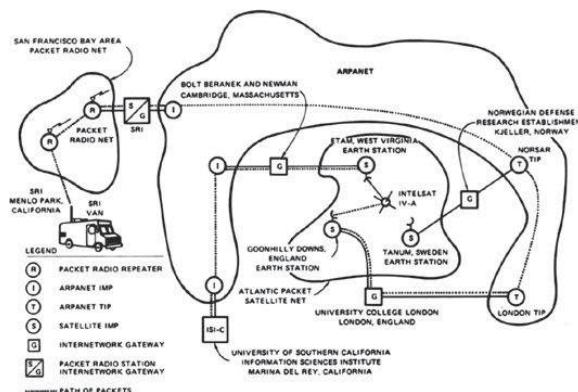
Taylor’s original vision for personal computing at Xerox Palo Alto Research Center (PARC) had always included networking the Alto machines in a local area network, or LAN. The networking technology needed to be cheap – Taylor’s goal was it should cost no more than 5 percent of the cost of the computers they were connecting. The technology also needed to be easily expandable, capable of linking hundreds of Altos. Researcher Bob Metcalfe (B.10.15) arrived at PARC in the summer of 1972 still smarting from the indignity of having his doctoral thesis turned down by Harvard as being “insufficiently theoretical.”³¹ While at Harvard, he had spent much of his time at MIT working on the ARPANET, and he had written up this very practical work for his thesis. At PARC, Metcalfe found several experimental networking projects in progress, but he believed that none of them would satisfy Taylor’s requirements. There was also a deadline looming: in his design for the Alto hardware, Chuck Thacker had left space for a yet-to-be-designed *network controller card*, a device to connect the Alto to a computer network, and PARC was nearly

ready to roll out the Alto. Metcalfe remembered reading a paper about an ARPA-funded networking project linking computers on the different islands of Hawaii. This Hawaiian network was called ALOHAnet and was designed by University of Hawaii professor Norman Abramson. Instead of sending electrical signals down phone lines, ALOHAnet used a radio network that sent digital signals through the atmosphere. Abramson had come up with a simple way to manage interference caused by two stations trying to transmit a message to the same receiving station simultaneously. If the transmitting stations did not get an acknowledgment of a successful receipt of the message from the receiving station, they were programmed to resend their messages after waiting different, random times. This delay ensured that their messages would not collide a second time. Metcalfe realized that this feature could be very useful for a LAN, in which there could be many computers trying to send messages at the same time. He also proposed joining the Altos together with a physical cable. Metcalfe thought of the Altos as sending their digital messages onto a wire that would merely act as a passive channel, much as the atmosphere served as an inactive medium through which radio signals traveled for ALOHAnet. He likened this use of the connecting wire to the way that physicists before Albert Einstein had assumed the presence of a *luminiferous* (light-bearing) *ether*, a substance through which light signals were supposed to travel. Metcalfe's first memo on his idea in May 1973 was titled "The ETHER network" and described his ideas for linking computers in a LAN.

In collaboration with Stanford graduate student David Boggs, who was then working part-time at PARC, Metcalfe tested out his ideas using a coaxial cable to connect the computers. A *coaxial cable* is a cable with an inner copper conductor surrounded by an insulator and a copper conducting sheath, all encased in plastic. The cable is called *coaxial* because the inner conductor and the outer conducting sheath have the same *axis* (center). The cable acts as a transmission line for radio-frequency signals. The cable connecting the computers was usually silent and so resembled the inert "ether." When a machine wanted to transmit a message, it sent a "wake up" bit onto the cable to alert the other machines. It then sent a packet with a destination address, a sending address, the message, and some checksum bits for error checking. If another machine sent a packet at the same time, resulting in a collision, both machines stopped sending and each waited a different random time before resending the message. Metcalfe and Boggs developed the electronics to make this happen and produced the first *Ethernet card* for the Alto, a device to attach the computer to the Ethernet. Adding new machines proved to be very easy and just required the end of a branch cable to be plugged into the main coaxial cable.

Metcalfe resubmitted his doctoral thesis to Harvard, still about packet-switching but now including a suitably theoretical analysis of ALOHAnet. The university accepted his thesis in June 1973. Boggs took a leave of absence from Stanford and joined PARC full-time. It would be another nine years before he received his Stanford doctorate. The patent for Ethernet was filed in March 1975 under the names of Metcalfe, Boggs, Thacker, and Butler Lampson. Metcalfe left Xerox PARC in 1979 to set up a new company called 3Com to produce Ethernet networking equipment. With the support of the Digital Equipment, Intel, and Xerox corporations, the DIX standard was published in 1980. It carried Ethernet

Fig. 10.16. The original conceptual plan suggested by Cerf and Kahn for a multinet system. This idea led to the birth of the “network of networks,” better known as the Internet.

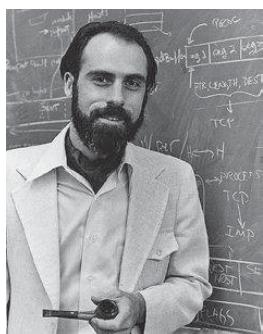


traffic at a rate of ten megabits per second. Adoption of Ethernet technology rapidly outpaced several competing technologies, and Ethernet became the dominant technology for building LANs.

The TCP/IP protocol and the Internet

Kahn and Cerf had first worked together in 1970 when Kahn was performing his network congestion experiments (B.10.16). By 1973, there were a number of different international packet-switching networks, and members of the network community had discussed what was needed to link them together. Typically the networks would have different interfaces, packet sizes, and transmission rates. Kahn and Cerf came up with the idea of a “gateway” computer. To connect the ARPANET to the gateway machine, the machine must just look like a host to the ARPANET IMPs. But to connect to another network, such as the ALOHAnet radio network, the gateway software must also allow the machine to look like a host on that network. Each network could still send messages to its own machines using its own protocols, but when data needed to be sent to another network, a new and universal networking protocol was needed – a set of rules that any computer connected to the network could follow to send or receive data.

In May 1974, Cerf and Kahn wrote a paper describing such a protocol (Fig. 10.16). They proposed that messages should be encapsulated in self-contained packets of information with the source and destination addresses written into the header, in much the same way that letters are put into envelopes and sent to their destination. The gateways would read the addresses on the “envelopes,” but only the receiving host would read the contents. This set of rules, called the Transmission Control Protocol (TCP), also made the assumption that the packet-switching network was intrinsically unreliable. In the construction of the ARPANET, it had been the IMPs that provided the reliability: now the focus of reliability shifted from the network to the communicating host computers. Cerf and Kahn led the discussion with the ARPANET participants and with the growing international networking community. This community included Donald Davies in the United Kingdom and Louis Pouzin in France, who was building a packet-switching network



B.10.16. Vint Cerf studied mathematics at Stanford and earned his PhD in computer science from UCLA in 1972. After receiving his doctorate, he became an assistant professor at Stanford and worked with Bob Kahn on the TCP/IP protocol for network interoperability, the basis for the modern Internet.

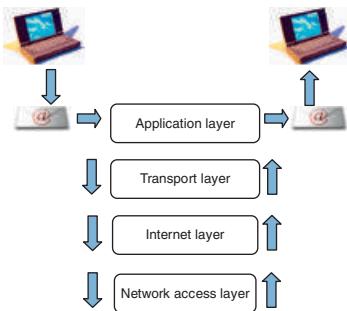


Fig. 10.17. The layered structure of the TCP/IP protocol.

called Cyclades. By 1975, the TCP specification had advanced sufficiently to be implemented concurrently at three sites – BBN, Cerf’s group at Stanford, and a group headed by Peter Kirstein at University College London. In October 1977, the effort reached a significant milestone when Cerf and Kahn demonstrated sending messages across three interworking networks: a packet radio network, a packet satellite network called SATNET, and the ARPANET.

The networking protocol took its final form in early 1978 when the part of the transmission protocol that deals with the routing of the packets was separated off as a discrete Internet Protocol (IP). Under the new TCP/IP protocol, TCP was now responsible for breaking up messages into datagrams, reassembling them at the other end, detecting errors, and resending lost messages. The IP was responsible for routing the individual datagrams. Jon Postel (B.10.17) later summarized the guiding principle for what should be in the IP: “I remember having a general guideline about what went into IP versus what was in TCP. The rule was ‘Do the gateways need this information in order to move the packet?’ If not, then that information does not go into IP.”³²

In the first chapter we mentioned the layered approach as one of the frequently occurring “mental tools” used in computer science. The TCP/IP protocol is a further illustration of this idea (see Fig. 10.17). TCP/IP consists of four layers. At the top there is the application that sends a message to another application running on a remote computer. At the bottom lies the actual physical wire or fiber where the message is translated into electric or light impulses. If the message is long, then it is sliced up into smaller pieces, each put into a separate envelope. Imagine if we sent an entire book to somebody one page at a time. On the Internet, as the message travels down it is put into a bigger envelope at each layer. We can think of the protocol as resembling *matryoshkas*, or Russian nesting dolls (Fig. 10.18). On the receiving end, the message travels upward, and at each layer the envelope is stripped off. Because the original message was sliced into pieces, the message must also be reassembled on the receiving end.

TCP/IP emerged as the dominant Internet standard after a long battle with a rival proposal for standardizing networking. The rival standard was called *Open Systems Interconnection* (OSI). It had been developed by the International Organization for Standardization (ISO), a world body that tries to establish uniform sizes and other specifications to ease the international exchange of goods. The U.S. government and European national governments all decreed that OSI was the official networking standard for the Internet. Similarly, major computer manufacturers like IBM, Digital, and Hewlett-Packard also adopted the OSI standard instead of TCP/IP. Amazingly, it was the popularity of the Unix operating system and Ethernet LANs in universities that turned the tide in favor of TCP/IP. Unix, developed at Bell Labs in the 1960s, is a popular operating system because it is powerful and stable, and can be installed on many different kinds of machines. Except for Microsoft Windows, nearly all major operating systems have some kind of Unix at their core. Bill Joy, later one of the founders of Sun Microsystems, received an ARPA grant to write the TCP/IP stack, a complete set of networking protocols, into the free Berkeley version of Unix. The first Sun machines were sold with Berkeley Unix including the TCP/IP networking software. With the worldwide popularity of Sun workstations and the increasing availability of Ethernet as a commercial product in the early

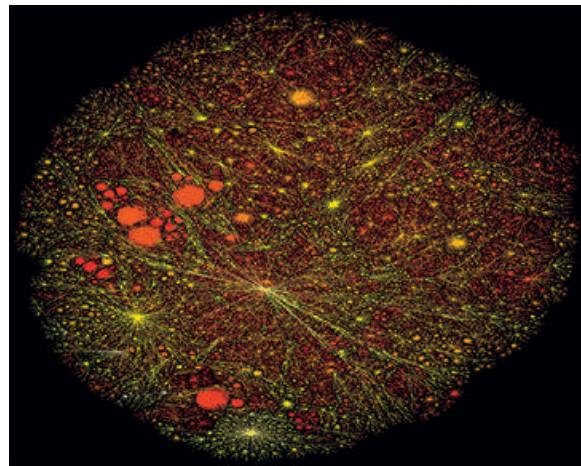


Fig. 10.18. Matryoshkas are Russian nesting dolls. Here they illustrate the concept of message encapsulation.



B.10.17. Jon Postel (1943–98), “the unsung hero of networking,” was the coordinator of the RFC discussion forum and later became chairman of the Internet Assigned Numbers Authority, in charge of assigning Internet numbers. Cerf wrote Postel’s obituary, which was published as RFC 2468.

Fig. 10.19. An Internet map in 2005 generated by tracing the packets from source to destination.



1980s, TCP/IP-based computer networking grew rapidly in universities. In the end, TCP/IP won out, showing that an open, grassroots process can sometimes prevail over an imposed official standard.

The Internet soon became much more than a research experiment for academic researchers (B.10.18). By the time the ARPANET was retired in 1990, commercial *Internet service providers* (ISPs) had begun to emerge, allowing business and the general public to connect to the Internet, usually for a monthly fee. For many years, the U.S. government, which funded ARPANET and its successor NSFNET, had tried to limit the Internet to research and educational uses. In 1995, however, the government removed the last restrictions on commercial use of the Internet. Traffic on the public Internet grew at more than 100 percent per year through the late 1990s. As of March 2011, the number of Internet users exceeded two billion, some 30 percent of the world's population (Fig. 10.19).

From copper to glass

The chapter began with an account of Chappe's early optical communication system, and it now ends with a discussion of how lasers and optical fibers



B.10.18. A group photo of Internet pioneers taken in 1994 to mark the twenty-fifth anniversary of the ARPANET. The photo was taken at a Christian Science Church in Boston in front of their world map. Left to right, front row: Bob Taylor, Vint Cerf, and Frank Heart; second row: Larry Roberts, Len Kleinrock, and Bob Kahn; third row: Wes Clark, Doug Engelbart, and Barry Wessler; fourth row: Dave Walden, Severo Ornstein, Truett Thach, Roger Scantlebury, and Charlie Herzfeld; fifth row: Ben Barker, Jon Postel, and Steve Crocker; last row: Bill Naylor and Roland Bryan.

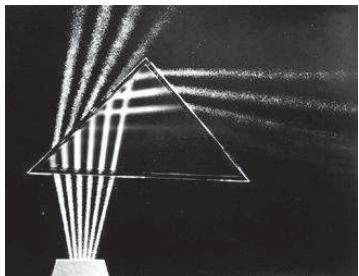


Fig. 10.20. Several rays of light striking a prism at a variety of angles. As can be seen, beyond a certain “critical” angle the light rays are entirely reflected and no light is transmitted through the prism. The ray on the extreme right is entirely reflected while the other rays show both transmitted and reflected beams.

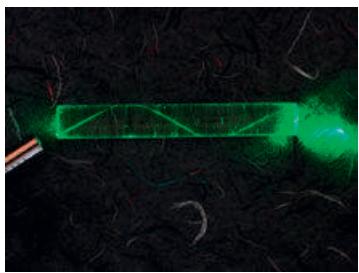


Fig. 10.21. Figure illustrating total internal reflection in an optical fiber.

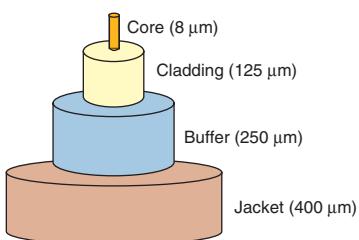


Fig. 10.22. Diagram of a modern fiber-optic cable showing the relative sizes of cladding and core.

have replaced copper wires for high-bandwidth communications and made possible the *broadband* Internet we see today. *Broadband* has various definitions, but the term now generally refers to high-speed data connections to the Internet. Optical fibers are thin “wires” of glass that transmit light using a phenomenon called *total internal reflection*. We can see how such reflection comes about by considering what happens when light travels from air into a block of glass. Because light travels more slowly in glass than air, the light changes direction, bending toward the vertical. This bending of light at a surface is called *refraction*. Now consider light traveling from glass to air, when the light bends away from the vertical. If we increase the angle at which we shine light on the glass-air surface, the transmitted ray emerges at an angle closer and closer to the surface. At an angle called the *critical angle*, the light just grazes the surface. If we increase the angle of incidence beyond this critical angle, all the light will now be reflected and no light will escape into the air (Fig. 10.20). This is the phenomenon of *total internal reflection* (see Fig. 10.21). It is this mechanism that allows light to be transmitted down a glass fiber and to travel around bends in the fiber. *Cladding* or covering (see Fig. 10.22) the core of the fiber with glass of a lower refractive index provides a way to bend or reflect inward light rays that strike its interior surface. This makes it possible for the light to travel long distances with little loss of intensity. Durable optical fibers in medical imaging devices like endoscopes carry light that enable doctors to examine the inside of the stomach and other organs (Fig. 10.23).

Optical fibers were fine for use in applications like endoscopes where the light only needs to be transmitted a few meters. But for transmission of light over the long distances required for telecommunications, scientists believed that the light in the fiber would lose too much intensity for optical fibers to be practical. Engineers use units called *decibels* to measure the energy loss of signals. In 1960, glass fibers had an *attenuation* (reduction in strength) of about one decibel per meter – meaning that about 20 percent of the light entering the fiber was lost in just traveling the width of a table. After traveling a hundred meters through a fiber, only one ten-billionth of the light would remain (see Table 10.1). A loss of ten decibels per kilometer would mean that a tenth of the power remained after a kilometer: a loss of one thousand decibels per kilometer – one decibel per meter – meant that almost no light remained. This gives an idea of the scale of the challenge. It was for this reason that Rudolf Kompfner, head of transmission research at Bell Labs, had dismissed optical fibers as a practical transmission technology in 1961. For optical fibers to become a viable communications technology, the attenuation in the fiber needed to be reduced to about ten decibels per kilometer.

Because of the attenuation problem, Bell Labs and almost all of the global telecommunications industry thought that the future for high-bandwidth communications would be based on *millimeter waveguides*, hollow pipes that could channel millimeter wavelength transmissions. But a few groups in the United Kingdom persisted in their research on optical fibers. At a meeting of the British Association for the Advancement of Science held in Southampton, England, in 1964, Alec Gambling (B.10.19), a professor in the Department of Electronics at the University of Southampton, suggested that glass fibers should be investigated “not because they looked at all promising, but Sherlock Holmes like, they seemed to be the least unlikely possibility to pursue.”³³ The real

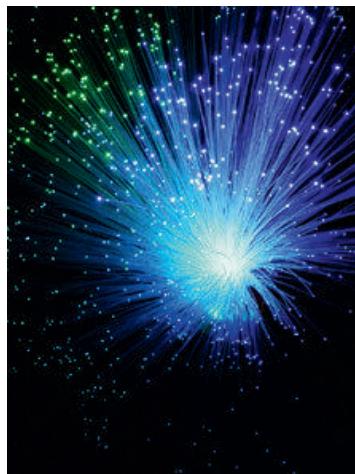
Table 10.1 Decibel table (the values of decibel loss are negative).

Decibel loss	Output/Input signal ratio
1	0.79
2	0.63
10	0.1
20	0.01
30	0.001
40	0.0001
50	0.00001
60	0.000001
70	0.0000001
80	0.00000001
90	0.000000001
100	0.0000000001

stimulus to optical fiber communication was a classic paper written by Charles Kao and George Hockham in 1966, working at ITT Corporation's Standard Telecommunication Laboratories (STL) in the United Kingdom (B.10.20). They had made a very careful study of the losses in various types of glasses and concluded that the high attenuation rates were due to impurities in the glass. The 1 April 1966, issue of *Laser Focus World* magazine noted:

At the IEE [Institute of Electrical Engineers] meeting in London last month, Dr. C. K. Kao observed that short-distance runs have shown that the experimental optical waveguide developed by Standard Telecommunications Laboratories has an information-carrying capacity ... equivalent to about 200 TV channels or more than 200,000 telephone channels. He described STL's device as consisting of a glass core about three or four microns in diameter, clad with a coaxial layer of another glass having a refractive index about one percent smaller than that of the core.... According to Dr. Kao, the fiber is relatively strong and can be easily supported. Also, the guidance surface is protected from external influences ... [and] the waveguide has a mechanical bending radius low enough to make the fiber almost completely flexible. Despite the fact that the best readily available low-loss material

Fig. 10.23. Light conducted by a bundle of optical fibers. The core of the fiber is only a few microns wide. A typical human hair is about fifty microns wide, so optical fibers are about a tenth of the diameter of hair.



B.10.19. Alec Gambling joined the Electronics Department at the University of Southampton in 1957. The department at Southampton had been founded in 1947 by Eric Zepler, a German refugee who had previously been head of radio receiver design at Telefunken. Gambling started his research in lasers as a potential source of high-frequency carrier waves for communications. After the classic paper on optical fibers by Charles Kao and George Hockham, Gambling received a research contract from the United Kingdom's Ministry of Defence and started research on manufacturing low-loss fibers. Gambling's group invented the method of Chemical Vapor Decomposition, which is still the most widely used method of optical fiber fabrication worldwide.

has a loss of about 1000 dB/km [decibels per kilometer], STL believes that materials having losses of only tens of decibels per kilometer will eventually be developed.³⁴

At this time, the British Post Office operated the British telephone network. After Kao's and Hockham's paper, Frank Roberts at the Post Office Research Station at Dollis Hill in the United Kingdom started a research program to reduce fiber losses. The third group researching fibers in the United Kingdom was Gambling's team at the University of Southampton. One of Gambling's research students, David Payne, built a drawing tower to make optical fibers. A *drawing tower* is the apparatus where a block of molten glass called a *preform* is *pulled* into a long thin fiber with the desired width and thickness. Gambling and Payne were able to reduce losses from thousands of decibels per kilometer to about 140. At about the same time, Charles Kao at STL made careful measurements in different types of glasses and concluded that the high purity of commercially available silica made it a good candidate material for optical communication fibers. After Bell Labs heard about these results from the United Kingdom, work on optical fibers restarted in earnest. It took only four years for the global research community to reach Kao's goal of creating an optical fiber with less than twenty decibels of light loss per kilometer.

Most groups were trying to purify the typical compound glasses used for standard optics, which were easy to melt and draw into fibers. At the Corning Glass Works in Corning, New York, Bob Maurer, Donald Keck, and Peter Schultz started with fused silica, a material that can be made extremely pure but has a high melting point and a low refractive index (B.10.21). By carefully adding controlled amounts of impurities called *dopants* they were able to raise the refractive index of the silica core to be slightly higher than the cladding without increasing the attenuation significantly. In September 1970, the Corning team announced the manufacture of a fiber with attenuation below twenty decibels per kilometer but gave no details of the manufacturing process. By 1972, the Corning team had managed to reduce losses to only four decibels per kilometer. Because the Corning group provided little information about how they had achieved their results, it was difficult for other groups to verify these claims. At a 1974 conference in Brighton, England, Gambling and Payne from Southampton announced their discovery of a new method of fiber fabrication



B.10.20. Charles Kao was awarded the Nobel Prize in physics in 2009 for “groundbreaking achievements concerning the transmission of light in fibers for optical communication.”³⁵ He was born in Shanghai in 1943 during World War II and, with his family he escaped by boat from mainland China to Hong Kong in 1948. He went to England in 1952 to study electrical engineering at what is now the University of Greenwich, London. After graduating, Kao stayed in the United Kingdom to work for Standard Telephones and Cables. In 1960 he was offered the opportunity to join STL research center and at the same time study for a PhD at University College London. At STL, Kao was asked to explore optical communication technologies and became a passionate advocate for optical fibers. In those days, glass fibers lost far too much energy in the light transmission through the glass. Kao showed that these losses were a result of impurities in the glass and speculated that fibers could be made with loss levels low enough to transmit signals long distances.



B.10.21. Optical fiber pioneers from Corning Glass Works: from left to right, Donald Keck, Bob Maurer, and Peter Schultz. The three developed the first low-loss optical fiber in 1970 after hearing of Charles Kao's goal of a loss of only twenty decibels per kilometer, equivalent to 1 percent of the light remaining after transmission through a one-kilometer glass fiber.

using *chemical vapor deposition* (B.10.22). This is a chemical process that deposits a thin coating on the silica and allows the creation of preforms with a specific refractive index profile. Simultaneous publications from the Southampton group and Bell Labs then gave details of a feasible method of manufacturing doped optical fibers. Using this method, Gambling and Payne were able to make fibers with a loss of only 2.7 decibels per kilometer. Although British Telecom became the first phone company to commit to optical fiber, it was the decision of MCI Communications Corporation to use optical fiber to build a nationwide long-distance communication network in the United States that really opened the floodgates. In 1982, MCI ordered one hundred thousand kilometers of fiber from Corning and took on the battle with the dominant AT&T. By the mid-1980s, the U.S. government had loosened many restrictions that limited competition in the long-distance telephone industry in the United States, creating a very competitive market for fiber networks.

The last piece of the puzzle arrived in 1987 when Payne and a team at the University of Southampton discovered a way to amplify the optical signal in the fiber. Up to then, when the optical signal grew too weak for further transmission, the signal had to be converted from optical to electrical, amplified, and converted back again to an optical signal. By adding a small amount of the element erbium to the core of the fiber, Payne and his colleagues at Southampton demonstrated that the signal could be amplified using a semiconductor laser as a *pump*, a device that boosts or amplifies a signal (Fig. 10.24). The first commercial product was produced in 1990 by Pirelli in the United Kingdom, in collaboration with the Southampton team. The first transatlantic fiber cable had a capacity of eight hundred telephone circuits, and the cost per circuit was \$30,000; only ten years later, transatlantic cables with fiber amplifiers had a capacity of six hundred thousand circuits at a cost per circuit of only \$500. In this way, fibers and *erbium-doped fiber amplifiers* (EDFAs) have revolutionized telecommunications and computer networks. Kao's original vision for optical fiber communications has come true: "If you really look at it, I was trying to sell a dream.... There was very little I could put in concrete to tell these people it was really real."³⁵

One postscript to this story is the curious fact that although the United States led the world in developing lasers, AT&T and Bell Labs virtually ignored the idea of fiber-optic communications. Equally surprising is how a small university research group at the University of Southampton with very limited funding not only competed with all the major telecommunication research laboratories but also made a string of fundamental discoveries. Payne believes

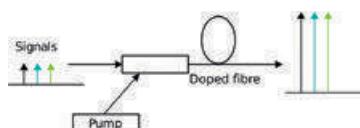
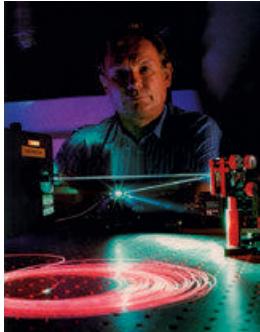


Fig. 10.24. Schematic representation of the working of the erbium-doped fiber amplifier (EDFA).



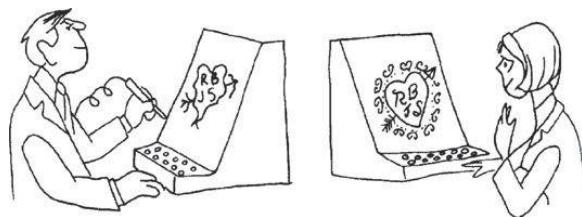
B.10.22. David Payne is director of the Optoelectronics Research Centre at the University of Southampton in the United Kingdom. He was a student of Alec Gambling's and built one of the world's first optical fiber drawing towers during his research for his PhD thesis. Payne later led the team that discovered the erbium-doped fiber amplifier in 1987. With Emmanuel Desurvire of Bell Labs, Payne was awarded the 1998 Benjamin Franklin Medal "for fundamental technical contributions and leadership critical to the successful development of the Erbium-Doped Fiber Amplifier and for the championship of this project."³²

that part of the answer is the ability of university research groups to pursue blue-sky research, scientific investigations without any obvious applications in the real world:

The optical amplifier was developed with no [research] contract. It did not have a proposal to do it, it did not have any milestones, it did not have any deliverables, we did it completely on the site. And I think that you'll find that most of the great developments in all areas of science are like that. So the present move towards focused and managed research might well be the wrong thing to be doing.³⁶

Key concepts

- Store-and-forward networks
- Centralized, hierarchical, and distributed networks
- Circuit switching, message switching, and packet switching
- Networking protocols and the ARPANET
- File Transfer Protocol and Telnet
- Wide area networks and email
- Ethernet and local area networks
- The Internet and the TCP/IP protocol
- Optical fibers for communication
- Erbium-doped fiber amplifier



"A communication system should make a positive contribution to the discovery and arousal of interests."³²

Women and telegraphy

The telegraph system required a large labor force to operate the transmitting and receiving machines (Fig. 10.25). This need provided new “high tech” jobs for women, many of whom could operate telegraph machines with great dexterity. In February 1846, only two years after Samuel Morse’s first successful demonstration of the electric telegraph, the Magnetic Telegraph Company opened an office in Lowell, Massachusetts, and hired a woman named Sarah Bagley as one of the first female telegraphers in the United States. Early in 1847, she was promoted to run the magnetic telegraph office in nearby Springfield, Massachusetts, but was understandably unhappy to learn that she earned only three-quarters as much as the man she replaced. This experience, together with her earlier experiences in the Lowell textile mills, led her to be an early advocate of women’s rights.

The *Illustrated London News* described the scene in a telegraph office in 1874:

It is a cheerful scene of orderly industry, and it is, of course, not the less pleasing because the majority of the persons here are young women, looking brisk and happy, not to say pretty, and certainly quite at home. Each has her own instrument on the desk before her. She is either just now actually busied in working off or in reading some message, or else, for the moment she awaits the signal, from a distant station, to announce a message for her reception. Boys move here and there about the galleries, with the forms of telegrams, which have been received in one part of the instrument-room, and which have to be signaled from another, but which first have to be conveyed, for record, to the nearest check-tables and sorting tables in the centre.³⁷



Fig. 10.25. The Central Telegraph Office London in 1874 employed 1,200 telegraphists of whom 740 were female and 270 boy messengers. Each day around 18,000 messages were transmitted. This engraving was produced to accompany an article in the *Illustrated London News*.

ARPANET gets funded

The story of how Bob Taylor got the funding to build the ARPANET is now the stuff of legend. In his office, Taylor had multiple terminals connecting him to different computers funded by ARPA at the various research centers across the country. Each terminal required a different login procedure, and no computer could “talk” to another. Frustrated by these incompatibilities, Taylor decided to act on Licklider’s idea. Without even writing a short memo about his plan, he went straight to the office of ARPA Director Charles Herzfeld. Fortunately, Herzfeld had seen the “multiple terminal problem” for himself and had also previously talked with both Licklider and Taylor about their ideas for interactive computing and networking. The gist of Taylor’s argument to Herzfeld was that because more and more researchers were requesting funds from ARPA to have their own expensive computers, it would be more cost-effective for ARPA to network the computers at the different sites so that researchers could share both the hardware and access each other’s results. Taylor suggested that ARPA should fund a small test network connecting these ARPA computers, initially with only four nodes, but then expanding to a dozen or more if it was successful. After just twenty minutes of discussion, Taylor left Herzfeld’s office with an extra million dollars in his budget to build such a network. The original program plan for the ARPANET talks about accessing time-shared computers and does not mention network survivability in case of nuclear attack.

II Weaving the World Wide Web

We should work toward a universal linked information system, in which generality and portability are more important than fancy graphics techniques and complex extra facilities. The aim would be to allow a place to be found for any information or reference which one felt was important, and a way of finding it afterwards. The result should be sufficiently attractive to use that the information contained would grow past a critical threshold.

Tim Berners-Lee¹

The hypertext visionaries

Vannevar Bush (B.11.1), creator of the “Differential Analyzer” machine, wrote the very influential paper “As We May Think” in 1945, reflecting on the wartime explosion of scientific information and the increasing specialization of science into subdisciplines:

There is a growing mountain of research. But there is increased evidence that we are being bogged down today as specialization extends. The investigator is staggered by the findings and conclusions of thousands of other workers – conclusions which he cannot find time to grasp, much less remember, as they appear. Yet specialization becomes increasingly necessary for progress, and the effort to bridge between disciplines is correspondingly superficial.²

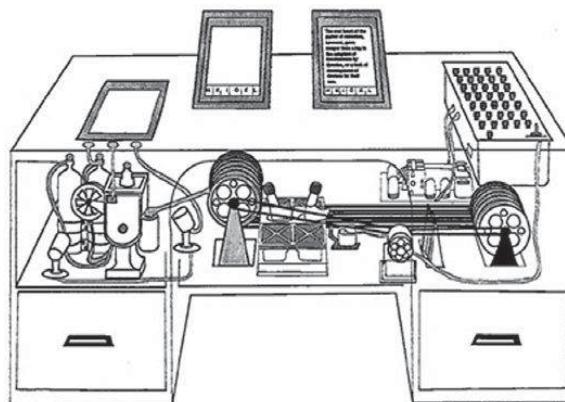
Bush concluded that methods for scholarly communication had become “totally inadequate for their purpose.”³ He argued for the need to extend the powers of the mind, rather than just the powers of the body, and to provide some automated support to navigate the expanding world of information and to manage this information overload. For this purpose, he introduced the idea of a new type of device:

Consider a future device for individual use, which is a sort of mechanized private file and library. It needs a name, and to coin one at random, “memex” will do. A memex is a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory.⁴



B.11.1. Vannevar Bush was born in 1890 in Massachusetts, attended Tufts College, and obtained his PhD in engineering from MIT in 1917. He joined the MIT Department of Electrical Engineering two years later and in 1927 started work on his Differential Analyzer – an analog computer for solving complicated systems of differential equations. One of his graduate students was Claude Shannon whose MIT master's thesis on using electrical relays to implement Boolean logic operations is one of the most-cited MIT theses. During World War I, Bush had been frustrated by the poor cooperation between the U.S. military and civilian scientists. When World War II broke out, he persuaded President Roosevelt to set up the National Defense Research Committee with him as chairman. Bush later said “if he made any important contribution to the war effort at all, it would be to get the Army and Navy to tell each other what they were doing.”⁸¹ Two of the most celebrated technological successes that were overseen by Bush were the Manhattan Project – that produced the atomic bomb – and the “proximity fuse” – a fuse inside an artillery shell that contained a miniature radar system so that the shell would explode when near the target. After the war, Bush’s report to President Truman – “Science: The Endless Frontier” – advocated federal funding of civilian basic research and resulted in the establishment of the National Science Foundation in 1950. He also wrote a second famous paper in 1945 that was titled “As We May Think.” It is surprising how many of the visionary ideas described in this paper are still relevant in the era of the World Wide Web, almost seventy years later.

Fig. 11.1. An illustration of the memex – a personal information system envisaged by Vannevar Bush to help users cope with the increasing flood of information.



Although the specific technology proposed by Bush to create the memex is now hopelessly out of date (see Fig. 11.1), his idea of recording “links” to represent associations between two pieces of information was the inspiration for today’s World Wide Web. By using such links, Bush thought we could mimic the working of the human mind in how it follows a trail of associations. This is how he imagined the memex machine working:

The owner of the memex, let us say, is interested in the origin and properties of the bow and arrow. Specifically he is studying why the short Turkish bow was apparently superior to the English long bow in the skirmishes of the Crusades. He has dozens of possibly pertinent books and articles in his memex. First he runs through an encyclopedia, finds an interesting but sketchy article, leaves it projected. Next, in a history, he finds another pertinent item, and ties the two together. Thus he goes, building a trail of

Fig. 11.2. The cover of Nelson's 1974 book *Computer Lib*. The book was published together with another book – *Dream Machines* – which was about the media-handling potential of computers.



many items. Occasionally he inserts a comment of his own, either linking it into the main trail or joining it by a side trail to a particular item. When it becomes evident that the elastic properties of available materials had a great deal to do with the bow, he branches off on a side trail which takes him through textbooks on elasticity and tables of physical constants. He inserts a page of longhand analysis of his own. Thus he builds a trail of his interest through the maze of materials available to him. And his trails do not fade.⁵

This was essentially the first description of *hypertext*, a text with interactive connections – *hyperlinks*, as we call them now – that give many options for moving between documents and allow a reader not to be restricted to just following a “linear” path through a document. It was this vision that inspired Doug Engelbart to start building his oN-Line System (NLS) in 1962. Engelbart’s NLS was the earliest working hypertext system, complete with mouse, and he first demonstrated it in public at the famous “Mother of All Demos” in San Francisco in 1968.

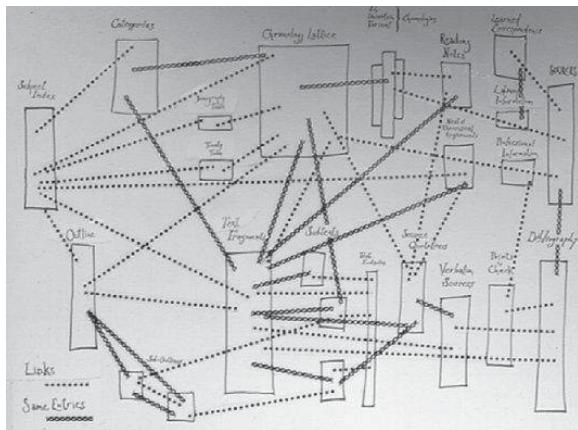
Bush’s ideas also inspired another visionary, Ted Nelson (B.11.2). Nelson was studying for a master’s degree in sociology at Harvard University in 1960 when he attended a course on computers. It was then that he conceived of building a software system that would allow nonsequential editing and reading of documents and permit the composition of compound documents that would display pieces of other documents, a concept he called *transclusion*. In a lecture at Vassar College in February 1963 Nelson described a system he called PRIDE, for *personalized retrieval indexing and documentary evolution*, that would enable the user to organize many types of research and personal notes. It was in this lecture that Nelson introduced the term *hypertext* (initially spelled with a hyphen) to describe his ideas about how to overcome the linear constraints of ordinary documents. The essence of hypertext was that it enabled nonlinear writing and reading, so that the reader could jump to another location in the text or move to a completely different document (Fig. 11.2).

Nelson refined his ideas over the next few years into an ambitious project he called Xanadu. He envisaged this as a system that supported two-way links; had unique and secure identifiers for users and documents; and provided a mechanism for tracking “micropayments” to authors for the use of parts of their writings. Nelson hired programmer Cal Daniels to produce a demonstration system of Xanadu in 1972. Two years later Nelson updated his vision to include networked computers and a repository for information that he called



B.11.2. Ted Nelson is both a visionary and a pioneer of many innovative ideas. He was born in New York in 1937 and has degrees from Swarthmore College in philosophy, from Harvard in sociology, and from Keio University in media and governance. Nelson has made important contributions to computer science with his ideas about hypertext – text that is linked to other information – and about new types of documents. Nelson also strongly supported personal computing with the rallying cry “Down with Cybercrud,” protesting the centralization of computers. In 1974, prior to the release of the Altair personal computer, he published the book *Computer Lib* with the subtitle *You Can and Must Understand Computers NOW*.

Fig. 11.3. Sketches of Ted Nelson's early global hypertext system from 1965. He also referred to this idea as a document universe or "docuverse." There are two types of links in this picture: the dotted lines represent normal hyperlinks, and the braided lines, representing links that point to quotations from other documents, Nelson called "transclusion" links. The system also introduced the idea of parallel text that enables us to see several related documents at the same time, as if we had several pages in front of us on a desk.



the *docuverse* (Fig. 11.3). However, it was not until 1998 that the first, still incomplete, Xanadu system was released and by then, the growth of the World Wide Web was already well under way. Although the present-day World Wide Web incorporates some aspects of his vision, Nelson calls Tim Berners-Lee's version of hypertext "precisely what we were trying to PREVENT – ever-breaking links, links going outward only, quotes you can't follow to their origins, no version management, no rights management."⁶

Vannevar Bush concluded his 1945 article with a surprisingly accurate vision of today's world of information. He accurately foresaw the emergence of such things as Wikipedia and social networks but totally missed the central role now occupied by Internet search engines:

Wholly new forms of encyclopedias will appear, ready made with a mesh of associative trails running through them, ready to be dropped into the memex and there amplified. The lawyer has at his touch the associated opinions and decisions of his whole experience, and of the experience of his friends and authorities. The patent attorney has on call the millions of issued patents, with familiar trails to every point of his client's interest. The physician, puzzled by a patient's reactions, strikes the trail established in studying an earlier similar case, and runs rapidly through analogous case histories, with side references to the classics for the pertinent anatomy and histology. The chemist, struggling with the synthesis of an organic compound, has all the chemical literature before him in his laboratory, with trails following the analogies of compounds, and side trails to their physical and chemical behavior.

The historian, with a vast chronological account of a people, parallels it with a skip trail which stops only on the salient items, and can follow at any time contemporary trails which lead him all over civilization at a particular epoch. There is a new procession of trail blazers, those who find delight in the task of establishing trails through the enormous mass of the common record. The inheritance of the master becomes, not only his additions to the world's record, but for his disciples the entire scaffolding by which they were erected.⁷

Fig. 11.4. Bricks-and-mortar libraries were the traditional way of doing research before the advent of the World Wide Web. This photograph shows the famous Reading Room at the British Library in London. People wanting to use it had to apply in writing for a reader's pass, which would be issued by the Principal Librarian.

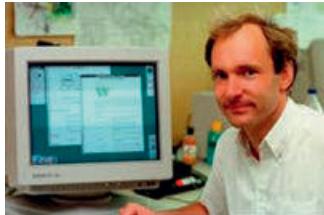


In June 1992, in an issue of the University of Minnesota *Wilson Library Bulletin*, librarian Jean Armour Polly wrote an article titled “Surfing the Internet.” In the article, she described how she could, from her home in New York, “surf” from server to server looking for information across the world:

Today I'll travel to Minnesota, Texas, California, Cleveland, New Zealand, Sweden, and England. I'm not frantically packing, and I won't pick up any frequent flyer mileage. In fact, I'm sipping cocoa at my Macintosh. My trips will be electronic, using the computer on my desk, communications software, a modem, and a standard phone line.⁸

Polly got the idea for the metaphor from a picture of an “information surfer” on her Apple Macintosh mouse pad. At the time, “surfing the Internet” was a very labor-intensive process. It meant explicitly downloading files from remote servers using the *file transfer protocol* (FTP). What finally made information surfing easy was the combination of two developments. One was the World Wide Web developed by Tim Berners-Lee and Robert Cailliau at CERN, the high-energy physics research laboratory in Geneva, Switzerland. The other was the easy-to-use Mosaic browser, the first web browser to gain popularity among the general public. Mosaic was created by Marc Andreessen, then still a student, and Eric Bina, a staff member at the National Center for Supercomputing Applications (NCSA) at the University of Illinois. The Mosaic browser included features such as *icons* (tiny pictures), *bookmarks* to store locations users might want to revisit, and a simple point-and-click method for finding, viewing, and downloading information that was appealing to people with little knowledge of computers.

Librarians were among the first people to see the impact of the Internet on how we access knowledge. Until recently, bricks-and-mortar libraries served as the temples of knowledge (Fig. 11.4), but with the arrival of the Internet and the web this has changed. We can now literally have all the books and all the information in all major libraries in the world at our fingertips. The invention of the World Wide Web and the technologies used to search it are the subject of this chapter.



B.11.3. Tim Berners-Lee graduated from Queen's College, Oxford in 1976 with a degree in physics. In 1980, at the CERN Laboratory near Geneva, Switzerland, he started to work on ideas for a novel “web” of information. His work was motivated by the need to develop a system that would provide fast access to manuals describing complex equipment, experiments, and other documentation used at CERN. Berners-Lee’s great contribution was to devise an engineering solution for combining the Internet and hypertext links into a powerful tool. *Time* magazine named him as one of the twenty most influential persons of the twentieth century and in 2003 he received a knighthood from the Queen.

Error 404 and the World Wide Web

In 1980, Tim Berners-Lee (B.11.3), a young physicist turned software engineer, accepted a temporary software consulting position at CERN, the famous European laboratory for particle physics near Geneva, Switzerland (Fig. 11.5).



Fig. 11.5. An aerial view of the CERN laboratory on the France-Switzerland border just outside Geneva, Switzerland. The large circle shows the location of the tunnel for the Large Hadron Collider, the world's largest particle accelerator. This machine made possible the discovery of the Higgs boson by particle physicists in 2012.

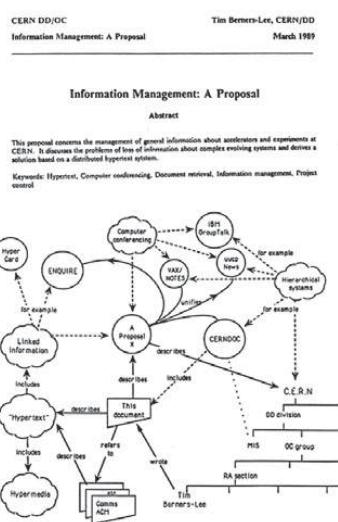


Fig. 11.6. Tim Berners-Lee's original 1989 proposal for a novel hypertext-based document management system at CERN. The figure shows circles with arrows, indicating documents, organizations, and people all connected by electronic links called hyperlinks.

CERN was then in the process of updating the control systems for its particle accelerators, devices that propel subatomic particles to high speeds, and Berners-Lee had been hired to help. For a temporary contract programmer, it was a major challenge to understand all the different components of the control system and to know who was responsible for each component. To help him keep track of all this information, Berners-Lee wrote a software program called *Enquire*. The name was a shortened version of the title of a how-to book called *Enquire within upon Everything*, a Victorian compendium of household advice that he remembered from his childhood. In his *Enquire* system, Berners-Lee could input a page of information about a person, a device, or a program, as he explained:

Each page was a “node” in the program, a little like an index card. The only way to create a new node was to make a link from an old node. The links from and to a node would show up as a numbered list at the bottom of each page, much like the list of references at the end of an academic paper. The only way of finding information was browsing from the start page.⁹

The program stored data in a much easier to use way than a traditional rigid hierarchical organization by allowing links to different paths of information. It had two types of links, an “internal” link within a file and an “external” link that could jump between files. The external link went in only one direction, which was important to avoid the problem of many people linking to the destination page and the owner having to store many thousands of return links. This initial program did not run on a network of computers but on a stand-alone computer.

Berners-Lee left CERN after about six months but returned to their Computing and Networks Division four years later. By this time, the researchers working on large particle physics experiments routinely networked their computers together, connecting the scientists working at CERN with one another and with their home institutions. After a year or so mulling over the new networked environment at CERN and his previous experiences with *Enquire*, Berners-Lee decided that a new type of “document management system” was needed that would essentially be a version of hypertext that operated over the Internet. In order that users would not be required to obtain access permissions from other users, the system would need to be completely decentralized with no center of control and with no one keeping track of all the available links. This decentralization was important because Berners-Lee was aware that this was the only way that the system could scale to accommodate thousands and even millions of users. Furthermore, he said, “the act of adding a new link had to be trivial”¹⁰ – because the easy addition of links was vital if his “web” of links was to spread around the world. In late 1988, Berners-Lee talked to his boss, Mike Sendall, who encouraged him to write up a proposal that would establish his idea as a formal project (Fig. 11.6).

At that time, Vint Cerf and Bob Kahn’s TCP/IP networking protocol for the Internet was not well established in Europe. Nevertheless Berners-Lee chose to adopt their protocol because the particle physics community used and loved the Unix systems that all supported TCP/IP for network communications. In March 1989, he gave a version of his proposal to Mike Sendall and Sendall’s



B.11.4. Official CERN photo of Peggie Rimmer, the manager in CERN's Documentation and Data Division who was responsible for hiring Tim Berners-Lee back to CERN in 1984. She, together with her husband, Mike Sendall, and the Documentation and Data Division director, David Williams, provided the necessary "air cover" from the physicists to allow Tim Berners-Lee to invent the web.



B.11.5. Robert Cailliau was born in Tongeren, Belgium in 1947 and has degrees in electrical and mechanical engineering from the University of Ghent and in computing from the University of Michigan. Cailliau was a co-author, with Tim Berners-Lee, of the 1990 version of the project proposal to CERN management for the World Wide Web.

boss, David Williams. Williams had the difficult job of defending his Data and Documentation Division – renamed in 1990 as Computing and Networks Division – from accusations by physicists that the division was wasting money on exotic computer science research that could better be spent supporting physics experiments (B.11.4). Berners-Lee's networked hypertext project was therefore only "informally" allowed to proceed. In 1990, with Williams deliberately turning a blind eye to the project, Sendall authorized the purchase for Berners-Lee of a new NeXT computer, a high-end workstation developed by Steve Jobs. Sendall told Berners-Lee, "When you get the machine, why not try programming your hypertext thing on it?"¹¹

Berners-Lee was now faced with the uphill task of convincing the CERN scientists, who were extremely busy and already highly proficient in the use of computers, that his idea of a global hypertext system was both exciting and useful. First he needed a name. After debating possibilities such as "Information Mesh" and "Mine of Information," he settled on the ambitious-sounding "World Wide Web." He was then fortunate to find the ideal evangelist to help him spread the message to the CERN community, the Flemish-speaking Belgian Robert Cailliau (B.11.5). Berners-Lee said later, "In the marriage of hypertext and the Internet, Robert was best man."¹² They began by trying to interest some of the companies that currently sold nonnetworked hypertext systems. Even Electronic Book Technologies, Inc., in Rhode Island, established by the legendary computer scientist Andy van Dam from Brown University, who had earlier collaborated with Ted Nelson, rejected Berners-Lee's proposal. Van Dam instead insisted on a centralized link database so that there would be no broken links. By contrast, Berners-Lee envisioned a truly dynamic system:

I was looking at a living world of hypertext, in which all the pages would be constantly changing. It was a huge philosophical gap. Letting go of that need for consistency was a crucial design step that would allow the Web to scale.¹³

He therefore started to write his own "web client" program that would allow the creation, editing, and browsing of hypertext pages. To identify the links in the documents, he developed a simple language called *HyperText Markup Language* (HTML). HTML used labels called *tags* to indicate where there were links to other documents. Publishers had used similar tagging schemes for many years to specify how documents should be formatted for typesetting.

The original HTML suggested by Berners-Lee contained only a dozen tags (see Fig. 11.7). Over the years, new tags were introduced for embedding images, multimedia, and scripts, so that the number of tags is now close to a hundred. To specify document addresses down to the specific file on a specific computer, Berners-Lee came up with the idea of a *universal resource identifier* (URI). The URI consisted of the computer server name followed by the directory path and file name of the document. For example, the address <http://info.cern.ch> specified the original CERN website, which contained information with links to other documents and sites (Fig. 11.8). The first four letters tell the browser which protocol to use to find the document. Berners-Lee introduced the *hypertext transfer protocol* (HTTP), a set of instructions that specified how a computer could communicate with other computers over the Internet to get to the desired content at a remote

```

<HTML>
  <TITLE>
    A sample HTML instance
  </TITLE>
  <H1>
    An Example of Structure
  </H1>
  Here's a typical paragraph.
  <P>
  <UL>
    <LI>
      Item one has an
      <A NAME="anchor">
        anchor
      </A>
    <LI>
      Here's item two.
  </UL>
</HTML>

```

Fig. 11.7. Example of a simple text with bracketed codes indicating the document elements in HTML from the June 1993 specification.

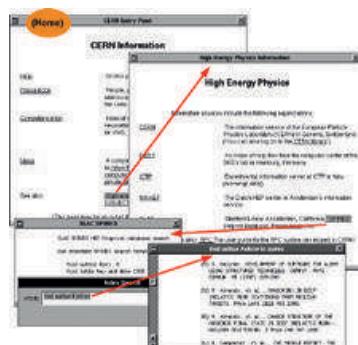


Fig. 11.8. Screenshot from the first text-based browser developed at CERN.

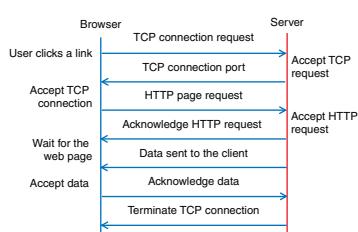


Fig. 11.9. The HTTP is a sequence-response procedure. The figure illustrates the sequence of messages between the browser and the server when the user clicks on a hyperlink.

site. This protocol requests a computer called the *web server* at the remote site to upload the requested web page to the user's browser for viewing (Fig. 11.9).

In November 1990, Nicola Pellow, a student visiting CERN from Leicester Polytechnic (now De Montfort University) in England wrote a web browser called a *line-mode browser* that worked on almost all computer terminals. Her browser enabled Berners-Lee's small team to release the World Wide Web program to people at CERN who had NeXT computers in March 1991. One of the key converts to the web was Paul Kunz, a visitor to CERN from the Stanford Linear Accelerator Center (SLAC) in Palo Alto, California. He was also a NeXT enthusiast. After Kunz returned to the United States, he worked with Louise Addis, the SLAC librarian, and their colleague Terry Hung to make SLAC's catalog of online documents accessible using the web. Berners-Lee announced this offering on 13 December 1991. The first web server in the United States, the SLAC home page, slac.html, was created six months later and was the first web server outside CERN (Fig. 11.10).

To evangelize the World Wide Web, Berners-Lee and Cailliau wrote up their work as a paper for the major Hypertext Conference sponsored by the Association for Computing Machinery that was scheduled to take place in San Antonio, Texas, in December 1991. It is part of the folklore of computer science that their paper was rejected! In spite of this rejection, they asked for permission to give a demonstration. This was not easy to arrange because, at that time, the conference provided no Internet connectivity to attendees. Cailliau had to call up a local university and arrange to use their dial-in service. As Berners-Lee says:

We were the only people at the entire conference doing any kind of connectivity.... At the same conference two years later ... every project on display would have something to do with the Web.¹⁴

The value of the web increased rapidly as more sites put up web servers and made their content available to other computers using the HTTP protocol. The real breakthrough happened when images started to appear on web pages (Fig. 11.11). In the summer of 1992, David Williams suggested that Berners-Lee should take sabbatical leave from CERN to visit the United States. Berners-Lee spent time at MIT (B.11.6) on the East Coast and at Xerox PARC on the West Coast promoting his ideas. While he was in the San Francisco area, he visited Paul Kunz and Louise Addis at SLAC but also took the time to pay homage to another Bay Area resident, Ted Nelson, who was the original inventor of hypertext.

By the end of 1992, the CERN team had a list of about thirty web servers, mainly in Europe, but also a handful based in the United States. NCSA at the University of Illinois, Urbana, was one of these U.S. sites. Traffic to the first web server at CERN was growing rapidly, with the number of daily *hits* – page views – doubling every three to four months. We can now see the relevance of “Error 404” in the title of this section. The error 404 message appears whenever a web client follows a link to a server but either the server is no longer there or the server is unable to find the requested page. The user typically receives an error message saying “Error 404 – Page Not Found” when attempting to follow such a broken or dead link. Berners-Lee’s great insight was to realize that such

The Computing Universe

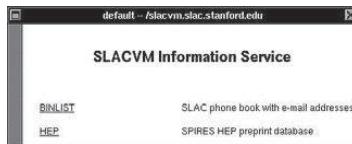


Fig. 11.10. The first U.S. website was set up at SLAC in 1991. Other physics laboratories around the world soon followed their example. These early web servers enabled the physics community to share documents and results of experiments.



Fig. 11.11. A picture of the CERN amateur singing group *Les Horribles Cernettes* was uploaded to the web by their fan Tim Berners-Lee on 18 July 1992, to test an early version of his software. It is believed to be the first photo posted to the web.



B.11.6. Michael Dertouzos (1936–2001) was director of the Laboratory of Computer Science at MIT. He played a crucial role in working with Tim Berners-Lee in establishing the World Wide Web Consortium (W3C) which oversees the development of web standards. In his book *The Unfinished Revolution* he outlined his vision for “human centric computers.”

broken links had to be tolerated if the World Wide Web was to scale up and be truly global and dynamic.

The web browser developed by Berners-Lee was specific to the rather uncommon NeXT workstations, and also required new windows to be opened as the user surfed from site to site. Marc Andreessen and Eric Bina at NCSA wanted to give users a much simpler experience and have pictures and text displayed in a single screen as a “mosaic” – that is, an assembly of small pieces of information. Their Mosaic browser was launched in April 1993 and tens of thousands of copies were downloaded from NCSA within the first few weeks (Fig. 11.12). Importantly, NCSA also developed versions of Mosaic for Windows and the PC so that their browser would reach the largest possible audience. From one hundred hits per day in the summer of 1991, by the summer of 1993 the CERN web server was receiving ten thousand hits per day (Fig. 11.13). The exponential growth of the web was now under way in earnest: from about fifty websites at the end of 1992, there were more than ten thousand sites by the end of 1994. The Mosaic browser was an important factor in fueling this growth. Another was CERN’s declaration, at the end of April 1993, “that CERN agreed to allow anybody to use the Web protocol and code free of charge, to create a server or browser, to give it away or sell it, without royalty or any other constraint.”¹⁵ This declaration was reassuring to industry and paved the way for the development of web-based e-commerce, business conducted over the Internet.

After graduating from the University of Illinois, Andreessen moved to Silicon Valley, California’s center of high-technology industries, in December 1993. He was keen to find some way to commercialize his work on the Mosaic web browser. With Jim Clark, one of the original founders of the computer manufacturer Silicon Graphics, Inc., Andreessen established the Mosaic Communications Corporation (later Netscape Communications Corporation) (B.11.7). Their first move was to hire the core Mosaic development team from NCSA, and their first browser, Netscape Navigator, was released in December 1994 (Fig. 11.14). Their browser worked with Unix, Windows, and Macintosh systems and was released as a download over the Internet. By making the software free for noncommercial use, Netscape Navigator rapidly became the *default browser* for the web, the browser that launched automatically unless the user changed it.

The dot-com bubble and the browser wars

To understand the rise of e-commerce, it is necessary to understand something about encryption. The science of cryptography dates back to ancient times and consists of techniques for *encoding* the information in a message so that it can only be *decoded* by its intended recipient. We discuss the state of the art in cryptography in more detail in Chapter 12 – here we will assume that reliable and computationally manageable encryption methods exist.

Netscape set the stage for the emergence of e-commerce by providing a new security facility called the *secure sockets layer* (SSL). This offered businesses the option of using an *encrypted* (coded), secure channel for users to send their credit card information over the Internet. The SSL makes the whole business of encryption invisible to the user. By 1994, with Netscape’s browser offering SSL

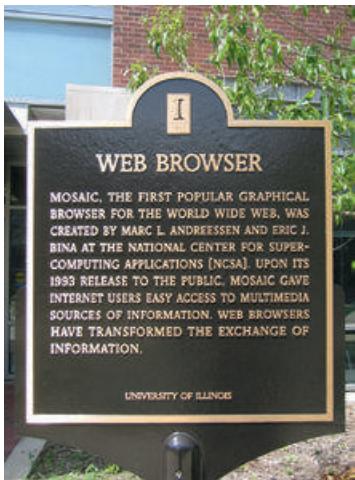


Fig. 11.12. Plaque commemorating the creation of the Mosaic web browser at NCSA at the University of Illinois, Urbana-Champaign.

encryption, all the elements were in place for e-commerce, or online retailing (a.k.a. dot-com), to take off.

To understand the significance of the term *dot-com*, we need to understand the naming convention that determines *domain names* on the Internet. Domain names were introduced in the early days of the ARPANET to serve as easily remembered names for ARPANET resources. The easily remembered domain names corresponded to an unmemorable string of numbers that was the actual *Internet Protocol (IP)* address. IP is the method by which data is sent from one computer to another on the Internet, and an *IP address* is a sequence of numbers that specifies a particular computer on the network.

At first, the mapping of computer host names to numerical addresses was held on a computer in Doug Engelbart's group at the Stanford Research Institute. In 1983, the Internet Engineering Task Force, a group that develops and promotes Internet standards, introduced the Domain Name System, which automatically translates the names we type in our web browser into IP addresses. Today, the Internet Corporation for Assigned Names and Numbers (ICANN) manages the assignment of top-level domain names (Fig. 11.15). A domain name consists of two or more labels, separated by dots, such as *microsoft.com*. The label after the first dot is the top-level domain, in this case, *.com*, signifying a commercial organization.

When the system was devised in the 1980s, there were two main groups of domains – a top-level country domain consisting of a two-letter abbreviation, such as *.uk* for the United Kingdom, and a top-level domain in the United States for seven types of organizations, such as *.com* for businesses. The other six were *.gov* for government, *.edu* for education, *.mil* for military, *.org* for organizations, *.net* for network, and *.int* for international. After the top-level domain comes the second-level domain and so on, as in *southampton.ac.uk*, where the second-level domain name *.ac* (equivalent to *.edu* in the United States) represents an academic organization in the United Kingdom, in this case the University of Southampton. In the domain name www.cern.ch, *www* signifies the web server at the CERN laboratory in Switzerland, which is specified by the *.ch* top-level, country domain name.

Fig. 11.13. There was rapidly growing interest in the web in the early days from 1992 to 1994. This figure shows the exponentially increasing load on the CERN web server (note the vertical scale).

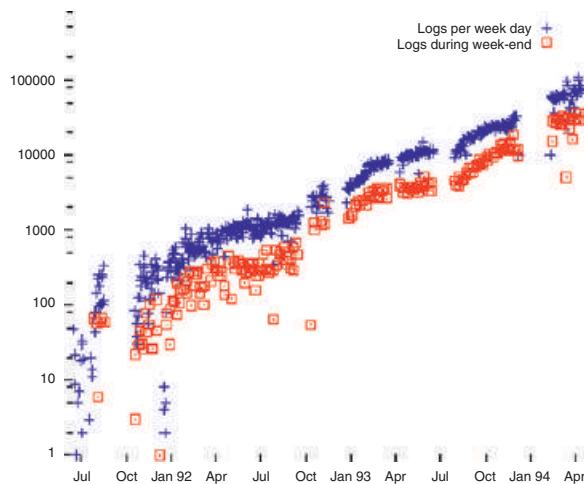




Fig. 11.14. First release of Netscape Navigator browser in 1994.



Fig. 11.15. The ICANN is a nonprofit organization responsible for assigning top-level Internet domain names.

The first commercial Internet domain name, symbolics.com, was registered in March 1985 by Symbolics, Inc., a computer systems company in Massachusetts. By 1992, fewer than fifteen thousand .com domains had been registered. All this changed in August 1995 when Netscape made an initial public offering (IPO) on the stock market. From an initial stock price of \$28 per share, the shares jumped to a peak of \$75 on the first day of trading. Netscape was now a company valued at more than \$3 billion even though it had yet to generate a profit. Sales for its Netscape Navigator browser were small but growing rapidly each quarter. By the end of 1995, the share price had risen to \$175 per share and investors were looking for other .com Internet companies that could show similarly spectacular growth.

One of the earliest and most enduring of the “dot-com” businesses is the online retailer Amazon (Fig. 11.16). Its founder, Jeff Bezos (B.11.8), was an investment analyst in New York when he saw a report in May 1994 describing the explosive growth of the World Wide Web. He recognized the potential of the web for online retailing and decided that selling books online would have a much lower overhead than traditional bricks-and-mortar bookstores. He left his job in Manhattan, moved to Seattle, and established Amazon in July 1994.

When Netscape’s IPO started the rush to invest in “dot-coms,” Internet start-ups sprung up offering everything from airline tickets and hotel rooms, such as *Lastminute.com*, to selling pet supplies, such as *Pets.com*. Throughout the second half of the 1990s, the dot-com bubble grew. A wave of investor enthusiasm pushed the stock value of Internet companies to greater and greater heights although most of the companies had not yet made any money. These new companies seemingly made a virtue of not having a conventional business plan showing a plausible path to a profitable future! A small group of financial analysts specializing in high-technology companies fueled the dot-com buying frenzy. Having seen the spectacular rise of Netscape, these Internet analysts talked up the value of these dot-com companies to unrealistic values as compared to traditional bricks-and-mortar companies. In February 2000, the inevitable happened: reality returned to the stock market and the shares of dot-com companies went into free fall. Amazon shares, for example, which had reached \$600 per share in 1999, fell to less than \$10 by the end of 2001.

Many of the dot-com companies disappeared without a trace, but Amazon continued to grow its business and reported a profit by the end of 2002, only one year later than Bezos’s original business plan had predicted. What of Netscape, the company that started the boom? Their dominant market share collapsed dramatically when Microsoft Corporation woke up to the potential – and threat – offered by the Internet and the web.

The story of Microsoft’s conversion to embracing the web and the Internet is long and complicated. One beginning was in September 1991 with the recruitment of James Allard, known within Microsoft as “J Allard.” Allard came from a Unix background and believed in the value of open *Application Programming Interfaces* (APIs). An API is a description of exactly how one program needs to ask another software program to perform a specific service. An open API is just an interface whose specification is freely available to the public so that any user can develop applications and services for a particular software program. It was clear to Allard that Microsoft urgently needed



B.11.7. The first CEO of Netscape Jim Barksdale (left) and the two founders, Marc Andreessen and Jim Clark.

Fig. 11.16. View inside an Amazon warehouse, which sorts products and packs and sends out orders for the online retailer.



an open API to link Windows to the Internet TCP/IP protocols. In the next year or so, Microsoft led a collection of companies in defining this interface, called the *Windows sockets interface* or “Winsock.” Microsoft officially endorsed the API in January 1992. Peter Tattam from the University of Tasmania in Australia released an open-source version. His software, called Trumpet Winsock, enabled users to connect to the Internet from Windows 3.0, which had no built-in TCP/IP support. Although few people paid Tattam for his software, Trumpet Winsock helped fuel the growth of the Internet by allowing millions of PC users to connect to the Internet for the first time.

Another step in Microsoft’s conversion to the web was caused by a snowstorm in Ithaca, New York. Steven Sinofsky, then Bill Gates’s technical assistant, was visiting Cornell University on a recruiting trip in February 1994. A snowstorm shut the airport, and he was forced to return to the campus. Things had changed since he had been a student there only seven years before. Many students now had their own PCs or Macs and campus computing resources could be accessed through the campus TCP/IP network. Students and staff routinely communicated by email, and use of the World Wide Web was growing rapidly. Still trapped in Ithaca, Sinofsky sent an email to Gates headed “Cornell is WIRED.”

Twice a year, Bill Gates used to take what he called a “Think Week,” spending several days in seclusion reading research papers and pondering the future of technology. Sinofsky’s email from Cornell triggered Gates’s famous “Think Week” memo in May 1995, which said, “The Internet is a tidal wave. It changes the rules. It is an incredible opportunity as well as incredible challenge.”¹⁶ It was a long memo and in it, Gates made many prophetic comments:

In this memo I want to make clear that our focus on the Internet is crucial to every part of our business. The Internet is the most important single development to come along since the IBM PC was introduced in 1981. It is even more important than the arrival of the graphical user interface (GUI).

The Internet’s unique position arises from a number of elements. TCP/IP protocols that define its transport level support distributed computing and scale incredibly well. The Internet Engineering Task Force (IETF) has defined an evolutionary path that will avoid running into future problems even as



B.11.8. Jeff Bezos, founder of Amazon, built the company steadily from its beginnings as a dot-com start-up in 1994 to its current position as the dominant, global, online retailer. He was inspired to resign from his job as an investment analyst in New York after seeing the explosive growth of the World Wide Web.



Fig. 11.17. The “Browser Wars” were a rivalry between Netscape Navigator and Microsoft’s Internet Explorer. As a joke, in the early hours of the morning, the Microsoft team celebrated the launch of IE 4.0 in 1997 by placing a giant e on the lawn in front of Netscape’s buildings in Mountain View, California.

eventually everyone on the planet connects up. The HTTP protocols that define HTML Web browsing are extremely simple and have allowed servers to handle incredible traffic reasonably well. All of the predictions about hypertext – made decades ago by pioneers like Ted Nelson – are coming true on the Web.

Amazingly it is easier to find information on the Web than it is to find information on the Microsoft Corporate Network. This inversion where a public network solves a problem better than a private network is quite stunning.

I think that virtually every PC will be used to connect to the Internet and that the Internet will help keep PC purchasing very healthy for many years to come.¹⁷

As a result of Gates’s memo, Microsoft dramatically changed course. In August 1995, at the same time as the company launched the Microsoft Network (MSN), a collection of Internet sites and online services, it also included a web browser called *Internet Explorer* in its release of Windows 95. It soon became clear that the vast majority of users preferred connecting to the free Internet rather than subscribing to a commercial consumer network like MSN or America Online (AOL).

Microsoft’s first browser, based on licensing the original NCSA Mosaic code, was fairly primitive. Over the next two years, Internet Explorer and Netscape Navigator battled for supremacy in the browser market, with each company releasing several browser upgrades each year (Fig. 11.17). By January 1998, Internet Explorer was not only the technical equal of Netscape Navigator but was also considerably more stable, with many fewer bugs. Because Internet Explorer was bundled free with the Windows operating system, it rapidly became the browser of choice for PC users. Netscape’s share price fell from its peak of \$175 per share to under \$15.

Before AOL acquired Netscape in 1999, Netscape released the source code for the browser and set up the Mozilla Foundation to manage its future development. The foundation describes itself as “a non-profit organization that promotes openness, innovation and participation on the Internet.”¹⁸ In 2004 Mozilla released the Firefox browser and, by 2007, it had gained a significant market share, despite the dominance of Internet Explorer and new entrants to the browser world, such as Safari from Apple and Chrome from Google.



Fig. 11.18. An artist’s impression of the millions of web pages.

Internet search and the PageRank algorithm

By the mid-1990s, the number and types of websites on the World Wide Web had grown enormously. In the early days of the web, users had to find out about “good” websites by word of mouth. Now, finding specific information on the web was like looking for a needle in a haystack – a user needed to search through an unorganized jumble of millions of web pages (Fig. 11.18). In 1994 two Stanford graduate students, Jerry Yang and David Filo (B.11.9), created a website called “Jerry’s Guide to the World Wide Web,” which was an alphabetical directory of interesting websites. Later that year, they renamed the website Yahoo! and set up a company that grew rapidly during the dot-com boom. In an attempt to keep up with the rapid growth of the web, Yahoo! employed teams of editors to help select the websites for the directory.

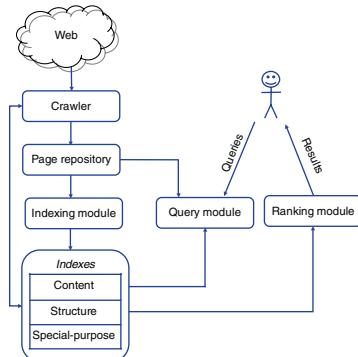


Fig. 11.19. Basic structure of a search engine showing the query-independent elements that respond to a user's query.

Other companies took a different approach by providing web *search engines*. These companies attempted to supply an index to the content of web pages at the different sites. A search engine works just like the index in a book in helping the reader look up a particular topic. By 1998 the leading search engine, with more than 50 percent market share, was AltaVista. Computer scientist Paul Flaherty at Digital Equipment Corporation's (DEC's) Network Systems Laboratory in Palo Alto had the idea of DEC building a web index. He recruited colleagues Louis Monier and Michael Burrows to write the software for what became the AltaVista search engine.

To create indexes for individual web pages, a search engine must first search out and capture these web pages (Fig. 11.19). This search is done with a *web crawler*, a piece of software that follows hypertext links to discover new web pages. The crawler sends out “spiders,” which are given explicit instructions on where to start crawling and what strategy to use in following links to visit new pages.

The web pages returned by the spiders now need to be indexed. The indexing software takes each new page, extracts key information from it, and stores a compressed description of the page in one or more indexes. The first type of index is called the *content index*. This directory stores information about the different words on the page in a structure known as an “inverted file,” which is similar to the index in the back of a book. Next to each term being indexed, the inverted file keeps information, such as the page numbers on which the term appears. We can now do single-word queries to find the relevant web pages. Of course, to efficiently handle more complex queries, we need to store more than just the page number for each word. We can add extra information, such as the number of times a word appears on a page, its location on the web page, and so on. A key advance made by AltaVista was also to store information about the HTML structure of the web page. By looking at the HTML tags on the page, we can identify whether the word being queried appears in the title, in the body of the page, or in the *anchor text*, the specific word or words used to represent the hypertext link. All of this indexed information is combined to deliver an overall “content score” for each web page to determine the most relevant page in answer to a query. It was this combination of content and structure information that had made AltaVista the leading search engine by 1998.

Modern search engines use more than just the content and structure to determine the best websites to return in answer to a user's query. It was the development of the *PageRank algorithm* by two Stanford graduate students,



B.11.9. David Filo and Jerry Yang founded the search company Yahoo! Inc. as Stanford graduate students. In 1994 they started compiling a directory of websites and extended the portal with a range of online services. By 1996 the company went public and became one of the landmark successes of the dot-com era. After the dot-com crash in 2000, the company suffered significant losses but Yahoo remains one of the household names of the Internet age, delivering online services to millions of customers.

Fig. 11.20. Aerial view of Google headquarters, the Googleplex, in Mountain View, California. The roofs on the buildings are covered with solar panels.



Fig. 11.21. Larry Page and Sergey Brin's first server at Stanford was encased in Lego blocks.



B.11.10. Eric Schmidt, Sergey Brin, and Larry Page (left to right) shown answering questions in 2008. While PhD students at Stanford, Sergey and Larry came up with the idea of ranking the web pages based on their link structure. They described their ideas in a much-cited paper “The Anatomy of a Large-Scale Hypertextual Web Search Engine.” After unsuccessfully trying to sell their ideas to AltaVista and Yahoo!, the two leading web search companies at that time, they set up Google, beginning in the traditional Silicon Valley garage.

Sergey Brin and Larry Page ([B.11.10](#)), that formed the basis for the success of Google ([Fig. 11.20](#)). The PageRank algorithm was a step-by-step procedure to calculate an “importance score” for each web page. Instead of just looking at the content and structure of web pages, Page and Brin analyzed the hypertext link structure. By combining their importance score from the link analysis with the content score from traditional indexes, Brin and Page developed a search engine that almost magically delivered the most useful websites to users.

Larry Page was intrigued by AltaVista’s information about the hypertext links and decided that analyzing the structure of the link data could be valuable. To do this, he started downloading as much as possible of the entire World Wide Web to his computer ([Fig. 11.21](#)). Meanwhile, Sergey Brin with his Stanford adviser, Rajeev Motwani, had been investigating the currently available search engines and directories. Page and Brin then teamed up on trying to accomplish Page’s goal of downloading as many web pages as possible and analyzing their link structure. Coming from an academic family, Page had the idea that the number of links to a web page was similar to the citation count of a scientific paper. The number of times other authors cite a paper is a significant indicator of the importance of the research described in the paper. However, Page realized that just counting the number of links pointing to a web page does not give the full measure of the importance of the page. Just as citations to a scientific paper from Nobel Prize recipients are more significant than citations by ordinary mortals, so too were links to a web page coming from an important or authoritative site. The journalist and author David Vise describes Page’s idea as follows:

All links were not created equal. Some mattered more than others. He would give greater weight to incoming links from important sites. How would he decide what sites were important? The sites with the most links pointing to them, quite simply, were more important than sites with fewer links.¹⁹

In a play on his last name, Page called his new algorithm PageRank.

How do you calculate the PageRank score of a given web page? If we assign an initial “authority” of 1 to each web page, we can calculate the accumulated authority of a given page by adding up the authorities of all the web pages that point to it. Unfortunately, the graph of web page links may contain a “cycle” – that is, by clicking on web links you eventually get back to the starting point (see [Fig. 11.22](#)). This makes it impossible to calculate an authority score for the sites in the cycle. To avoid this problem, Page and Brin introduced a “random surfer.” Imagine a surfer roaming the web following

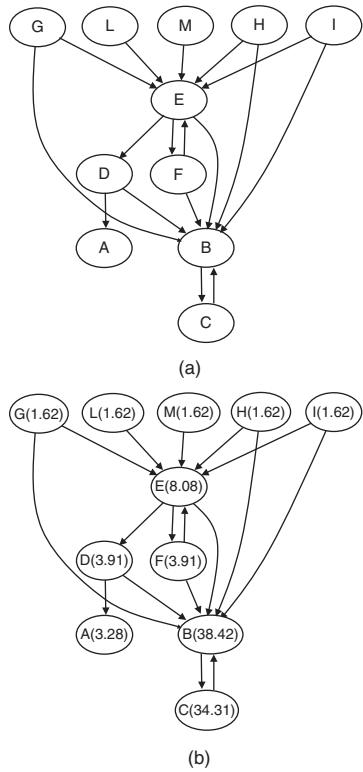


Fig. 11.22. (a) An example of a set of web pages – nodes – showing the hypertext links connecting the different pages with the direction of each link marked by an arrow. Web page A is known as a *dangling node* because this has no outgoing links. Pages B and C are said to form a *bucket*, a reachable strongly connected pair of pages with no outgoing links to the rest of the pages of the graph. (b) The importance of each page can be calculated using the PageRank algorithm. For this graph we have calculated the importance of each page using the *random surfer* method although in practice other methods are used. The scores have been normalized to add up to 100. Page C receives just one link but this is from the most important page. Page C ends up being more important than page E – which receives more links but all but one from pages G, H, M, L, and I, which all have no incoming links and therefore have the minimum importance.

hypertext links from page to page. When he or she arrives at a web page with multiple links, the surfer randomly chooses one of the links. We can now attempt to calculate the importance of each page by counting how many times the random surfer visits each site. To avoid the problem of *buckets* – cycles where the surfer can go around forever – Page and Brin introduced a “teleportation” probability. At each page the surfer visits, there is a chance that he or she does not follow one of the links from the page but instead jumps to an entirely new web page chosen at random. The surfer then uses this page as a new starting point for continued surfing. In addition, to avoid the problem of the surfer getting stuck in a so-called *dangling node*, a page with no outgoing web links, Page and Brin also allowed the surfer to jump to a random new page from such nodes. We can simulate the action of the random surfer on a network of web links and find out which pages the surfer visits most often. The importance score calculated in this way takes into account both the number of hyperlinks pointing to a web page and the authority of the linking web page. The method works in the presence of link cycles and dangling nodes and is able to deliver meaningful and stable values for each page’s importance.

Figure 11.22a shows a network of eleven web pages and their associated links. A computer simulation of the random surfer algorithm gives the PageRank score shown in Figure 11.22b. In practice, the computation of the PageRank score is not done by performing a computer simulation of the surfer’s travels. Instead, the problem can be formulated as solving a mathematical problem involving *sparse matrices*. A *matrix* is an array of numbers or symbols arranged in rows and columns, and *sparse* just means that the matrix has many elements that are zero. The matrices concerned are huge, with many billions of rows, but fortunately fast computational methods exist that can be used to find the PageRank scores.

Armed with the PageRank algorithm, by early 1997 Page had developed a prototype search engine that he called “BackRub” because it analyzed the incoming or “back” links to web pages to calculate a page’s importance. While other searches relied on the content and structure indexes, BackRub added the dimension of importance to rank pages in order of likely relevance. By the end of the year, Page and his officemate at Stanford, Sean Anderson, were brainstorming for a new name for the search engine. They decided on the name Google, thinking the word *google* meant a very large number. Page registered Google.com the same evening only to find out next day that the word they were thinking of was spelled *googol*, meaning the number 1 followed by one hundred zeros. With their search engine now being used by Stanford students and faculty, Page and Brin needed to beg and borrow as many computers as they could to keep up with both the growth of users and that of the web. Page’s dorm room became the first Google data center, crammed with inexpensive PCs (Fig. 11.23). Today Google has data centers around the world that run hundreds of thousands of servers (Fig. 11.24).

Having demonstrated the superiority of a search engine using the PageRank algorithm to calculate a web page’s importance, Page and Brin tried to interest AltaVista in buying the rights to their system. Paul Flaherty, one of the originators of the AltaVista search engine, was impressed with their link-based approach to page ranking. Nevertheless, Flaherty came back to them saying

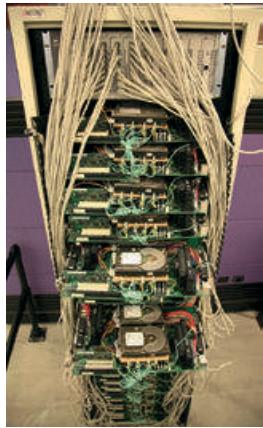
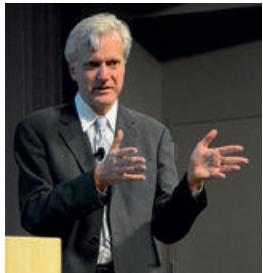


Fig. 11.23. The first Google production server rack from about 1998. The rack has multiple computer boards stacked one above the other.



B.11.11. Andy Bechtolsheim, one of the founders of Sun Microsystems, was Google's first investor.

that Digital was not interested. Page and Brin got the same lack of interest from other search-engine companies and from Yahoo!. However, Yahoo! cofounder David Filo gave them some valuable advice: take a leave of absence from Stanford and start their own business. By the summer of 1998, Page and Brin had decided to take Filo's advice. They had a meeting with Andy Bechtolsheim ([B.11.11](#)), one of the original founders of Sun Microsystems and now a Silicon Valley investor in start-up businesses. Despite his concerns about the lack of a viable business model for search engines in general, Bechtolsheim was impressed by Page and Brin and, without any discussion of stock distribution, he wrote them a check for \$100,000, made out to Google Inc. Page and Brin had to keep the check uncashed for two weeks until they had incorporated Google as a company and opened a bank account.

Google's rapid rise to dominance in web searching is chronicled in detail in *The Google Story* ([Fig. 11.25](#)) by David Vise. A company called Overture had pioneered search-related advertising, providing one of the first business models for search engines. Despite their initial reluctance to embrace an advertising model, Page and Brin decided to implement a variant of Overture's basic idea. To maintain the integrity of their free search service, they insisted on keeping their home page entirely free from advertisements and on distinguishing the free search results from what they called "sponsored links." Advertisers bid in an online auction for priority placement in these sponsored links, which appeared when users searched on specific search terms. Google only made money when a user clicked on one of the ads displayed. By the year 2000, Google was handling fifteen million searches per day compared to ten thousand only eighteen months before.

Google's early breakthrough undoubtedly owed much to the idea of including an importance score based on PageRank. However, *search engine optimization* (SEO) companies have now become big business. These SEO companies advise advertisers how to ensure that their web pages will appear near the top of search results. Some SEOs offer more than just good advice about how to increase the advertiser's content and importance score. They also try to manipulate the results using *web spam*, the practice of manipulating web pages

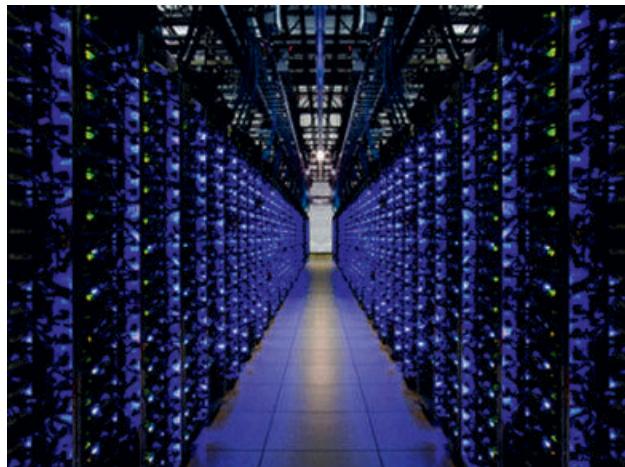


Fig. 11.24. One of the Google Data Centers with tens of thousands of computers installed on racks. The blue light indicates that the servers are running well.



Fig. 11.25. The cyber power stations of the twenty-first century. Steam is rising from the cooling towers of Google Data center in "The Dalles, Oregon."

deceptively so that search engines rank the pages higher than they would without any manipulation. Web spam can take a number of forms. *Term spamming* employs the term in question repeatedly on the web page, sometimes using white text on a white background so that the spam is invisible to human readers. *Cloaking* involves deceiving a web crawler by having different pages for normal requests than for spiders. The spider returns a clean web page without spam. Another way to influence the importance score is to automate the creation of a large number of different web pages with links to the customer's site. For all these reasons, modern search engines now include many other factors besides PageRank in arriving at their ranking of search results. For example, Harry Shum from Microsoft's Bing search engine team has said, "We use over 1,000 different signals and features in our ranking algorithm."²⁰

The social web and beyond

Darcy DiNucci, a consultant on information design, first introduced the term *Web 2.0* in 1999 in an article called "Fragmented Future":

The relationship of Web 1.0 to the Web of tomorrow is roughly the equivalent of *Pong* to *The Matrix*. Today's Web is essentially a prototype – a proof of concept. This concept of interactive content universally accessible through a standard interface has proved so successful that a new industry is set on transforming, capitalizing on all its powerful possibilities. The Web we know now, which loads into a browser window in essentially static screenfuls, is only an embryo of the Web to come.

The first glimmerings of Web 2.0 are beginning to appear, and we are just starting to see how that embryo might develop.... The Web will be understood not as screenfuls of text and graphics but as a transport mechanism, the ether through which interactivity happens.²¹

The term *Web 2.0* was later popularized by computer book publisher Tim O'Reilly at the first Web 2.0 conference, held in 2004. The term does not refer to an update of any particular technical specification but rather to the way in which software developers and users are now using the web, focusing on collaboration, user-generated content, and social networking. Web 2.0 applications allow users to interact and collaborate in new ways to create virtual communities. This development is visible through the growth in popularity of the online journals called *blogs* and of *wikis*, websites that allow users to make changes and add their own contributions.

The word *blog* is an abbreviation of *web log* and is usually an online diary recording the thoughts or actions of an individual (Fig. 11.26). Blogs are interactive in that, on most blogs, readers can leave comments and participate in an online discussion. The growth of blogging was accelerated by new, easy-to-use web publishing tools that did not require the user to have any knowledge of technologies such as HTML or FTP. By 2011, there were more than 150 million public blogs.

A *wiki* is a website that allows its users to interact with the site to add, modify, or delete content. The name *wiki* is a Hawaiian word meaning *fast* or *quick*. Ward Cunningham (B.11.12), inventor of the wiki, described it as "the simplest



B.11.12. Ward Cunningham invented the first collaborative *wiki*, a website that allows users to make changes and add their own contributions. *Wiki* is a Hawaiian word meaning fast or quick.

Fig. 11.26. Tony Hey's personal blog on e-Science.

The screenshot shows a blog post titled "Cyberinfrastructure Vision Moves Forward". The post discusses the definition of e-Science from an Educate report, mentioning the integration of supercomputing, sensors, data and information management, advanced instruments, visualization environments, and people, all linked together to advance and advanced networks to improve scholarly productivity and enable knowledge breakthroughs and discoveries not otherwise possible. It notes that while the definition is not explicitly stated, it seems like a pretty good definition of science.

On Wednesday this week, the Advisory Committee on Cyberinfrastructure (ACCII), a committee level in the National Science Foundation's Office of Cyberinfrastructure, met at NSF headquarters in Arlington, Virginia. The day was notable for its bone-chilling cold, and even more so for the significance of the meeting, during which each of six task forces relayed their "almost final" draft reports. These task forces had been set up last year by then director Ed Seidel to work with the research community in developing recommendations on how ACCII could best address NSF's 2007 "Vision for Cyberinfrastructure".

Disclaimer: The postings on this site are not on a and don't necessarily represent those of my employer or customers, partners, or sponsors.

online database that could possibly work.”²² He produced software that allowed users to interact with the wiki using a web browser. In some ways it is remarkable that public wikis work at all. For example:

Most people, when they first learn about the wiki concept, assume that a Web site that can be edited by anybody would soon be rendered useless by destructive input. It sounds like offering free spray cans next to a grey concrete wall. The only likely outcome would be ugly graffiti and simple tagging, and many artistic efforts would not be long lived. Still, it seems to work very well.²³

Wikis do sometimes get vandalized, but most users abide by the rules chosen collaboratively for the wiki’s governance. The most famous wiki of all is, of course, Wikipedia (B.11.13), the online encyclopedia created by its users. It sums up its policies and guidelines in five pillars:

- Wikipedia is an encyclopedia
- Wikipedia has a neutral point of view
- Wikipedia is free content
- Wikipedians should interact in a respectful and civil manner
- Wikipedia does not have firm rules

Although it undoubtedly has its faults, Wikipedia is a remarkable collaborative creation and makes a massive amount of content freely available.

Another important attribute of Web 2.0 is the ability of users to invent their own tags and to use these to bookmark photos and other material on the web. For example, Flickr allows users to tag their photos and use these tags for organizing and searching through their collection. Tagging is also used by virtual communities to create *folksonomies*, a grassroots way of classifying content based on user-generated tags or keywords that annotate and describe the information. Unlike a traditional hierarchical taxonomy, in a folksonomy all tags have more or less equal status. Finally, Web 2.0 offers the capability to create *mash-ups*, web pages or applications that allow users to combine data from multiple websites. It is common to make mash-ups with map data and to overlay different data sets such as houses for sale, the traffic status, and so on.



B.11.13. Jimmy Wales worked in the finance industry before starting the free web encyclopedia Wikipedia in 2001. Time magazine named him in its list of “The 100 Most Influential People in the World” in 2006.

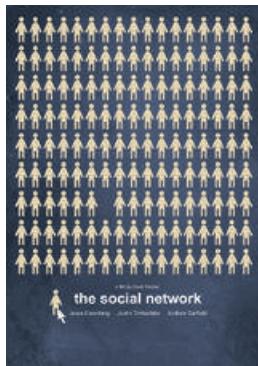


Fig. 11.27. The founding of Facebook was made into a movie called *The Social Network*. The marketing slogan for the movie was “You don’t get to 500 million friends without making a few enemies.”



Fig. 11.28. The Twitter service was introduced in 2006 and in 2013 had more than half a billion users. Twitter is an Internet text-messaging service that allows a maximum message size of 140 characters.

From these beginnings, new companies such as Facebook (B.11.14) and Twitter have now emerged. Facebook is a social networking site whose users can share personal updates, photos, and other information with their friends (Fig. 11.27). Twitter is a microblogging service that lets a person send brief text messages up to 140 characters in length to a list of followers. Twitter’s celebrity “tweeters” can attract millions of followers (Fig. 11.28).

Berners-Lee disagrees that Web 2.0 constitutes an essentially new vision for the web and dismisses it as marketing jargon:

Web 1.0 was all about connecting people. It was an interactive space, and I think Web 2.0 is, of course, a piece of jargon, nobody even knows what it means. If Web 2.0 for you is blogs and wikis, then that is people to people. But that was what the Web was supposed to be all along.²⁴

Instead, Berners-Lee is looking toward what he calls a “Semantic Web” in which machines can process and understand the actual data on the web:

The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation. The first steps in weaving the Semantic Web into the structure of the existing Web are already under way. In the near future, these developments will usher in significant new functionality as machines become much better able to process and “understand” the data that they merely display at present.²⁵

Industry has taken the first small steps toward such a semantic future. The search-engine companies Google, Microsoft, Yandex, and Yahoo! have agreed on a common vocabulary with which website managers can mark up their sites so that search engines can understand the nature of the content (schema.org). By inserting *microdata* – a set of machine-readable tags introduced with HTML5, the fifth revision of HTML – into the web page, and by using the agreed vocabulary, the website can specify the type of content on the site. For example does the website refer to *Casablanca* the movie or Casablanca the place? The search engine can now distinguish between the sites and better address the user’s query. These improvements are all part of the need to go beyond searching and better understand the user’s intent.



B.11.14. Mark Zuckerberg was still an undergraduate student at Harvard when he came up with the idea of using the web to link together a network of friends with a social networking website. This modest beginning has now evolved into the social networking company Facebook.

Key concepts

- Hypertext links
- World Wide Web: HTTP, HTML, and URLs
- Web browsers
- Domain names
- Internet search and PageRank
- Web 2.0 and the Semantic Web



Markup language goes hyper

During the twentieth century, typesetting advanced from “hot metal” to “cold type” composition, with photographic negatives replacing metal type as the source for making printing plates. By the 1960s, computer-driven phototypesetting machines had become commonplace, and it was against this background that in 1967, William Tunnicliffe, chairman of the Graphic Communications Association, proposed that a standard set of editorial markup instructions should be developed that could be inserted into a manuscript as directions to typesetters for printing. Charles Goldfarb at IBM adapted Tunnicliffe’s idea to develop a business system that would solve the problems of law firms in creating, editing, and printing documents. With his colleagues Edward Mosher and Raymond Lorie, Goldfarb created the *Generalized Markup Language* (GML) in 1973. GML was a set of rules and symbols that described a document in terms of its organizational structure and its content elements and their relationship. GML markups or tags described such parts of a document as chapters, important sections, paragraphs, lists, tables, and so on. By October 1986, the International Organization for Standardization (ISO) had adopted this language as an international standard called the *Standard Generalized Markup Language* (SGML). (The ISO is an international body that attempts to establish uniform sizes and other specifications to ease the exchange of goods between countries.) Berners-Lee wanted to make his new hypertext scheme as simple as possible but, at the same time, keep the goodwill of the global documentation community. He therefore deliberately designed HTML to look like a subset of SGML with only a small set of tags, inserted between angle brackets, as in <word>. Although Berners-Lee never thought that people would use a browser/editor to actually write web pages, the readability of HTML meant that many people did start writing their own HTML documents directly.

Emoticons

An emoticon is a pictorial representation of a facial expression that is meant to indicate the user’s mood at the time. The word is a portmanteau word made from the English words *emotion* and *icon*. In the online world emoticons are made up from regular keyboard characters such as :-) and :-(for happy and sad emotions of the sender. They are now often replaced by small images corresponding to these emotions such as ☺ and ☹. The use of emoticons can be traced back to the nineteenth century. The 1857 edition of the National Telegraphic Review and Operators Guide recorded the use of the number 73 in Morse code to express “love and kisses.”

Digital forms of emoticons were first proposed by Scott Fahlman to distinguish serious posts from jokes. His email to his colleagues read:

19-Sep-82 11:44 Scott E. Fahlman :-)

From: Scott E. Fahlman<Fahlman at Cmu-20c>

I propose that the following character sequence for joke markers:

:-(

Read it sideways. Actually, it is probably more economical to mark things that are NOT jokes, given current trends. For this, use

:-)

The actual “smiley face,” with two black dots for eyes and a black upturned curve for a mouth, both on a yellow circular background, was created by the artist Harvey Ball in 1963 (Fig. 11.29).

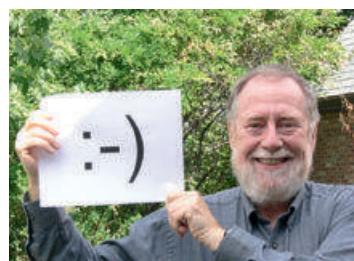


Fig. 11.29. (a) The computer scientist Scott Fahlman and his smiley emoticon. (b) The first smiley from 1963.

The sock puppet and the dot-com bubble

The fate of Pets.com is now a textbook lesson in the need to have a business plan that is grounded in reality. Pets.com was a company whose business was selling pet accessories and supplies to consumers directly over the web (Fig. 11.30). The company was launched in August 1998 and went public on the NASDAQ stock exchange in 1999. Despite the fact that its revenues were less than \$1 million in 1999, it spent nearly \$12 million of its start-up funding on a high-profile advertising campaign. This included a popular Pets.com sock puppet that was interviewed on the television show *Good Morning America* and an expensive TV commercial that ran during the 1999 Super Bowl. In the dot-com crash, Pets.com stock fell from more than \$11 per share in February 2000 to \$0.19 by 6 November 2000, the day it closed its doors and sold its assets to pay its debts.



Fig. 11.30. The Pets.com sock puppet was symbolic of the dot-com bubble. The company spent more than ten times its annual revenue on advertising and their popular sock puppet was even interviewed on the prime time TV show *Good Morning America*. The company went from a successful IPO to liquidation in less than a year.

12 The dark side of the web

When he later connected the same laptop to the Internet, the worm broke free and began replicating itself, a step its designers never anticipated.

David E. Sanger¹

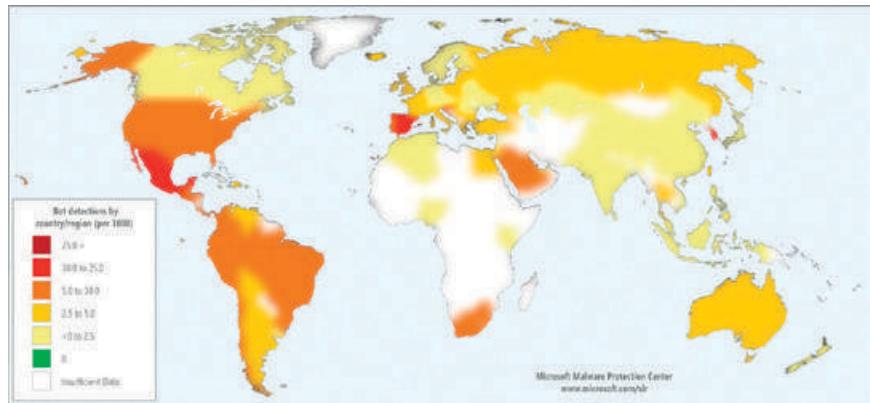
Black hats and white hats

As we have seen in [Chapter 10](#), the Internet was invented by the academic research community and originally connected only a relatively small number of university computers. What is remarkable is that this research project has turned into a global infrastructure that has scaled from thousands of researchers to billions of people with no technical background. However, some of the problems that plague today's Internet originate from decisions taken by the original Internet Engineering Task Force (IETF). This was a small group of researchers who debated and decided Internet standards in a truly collegial and academic fashion. For a network connecting a community of like-minded friends and with a culture of trust between the universities, this was an acceptable process. However, as the Internet has grown to include many different types of communities and cultures it is now clear that such a trusting approach was misplaced.

One example is the IETF's definition of the Simple Mail Transfer Protocol (SMTP) for sending and receiving email over the Internet. Unfortunately, the original SMTP protocol did not check that the sender's actual Internet address was what the email packet header claimed it to be. This allows the possibility of *spoofing*, the creation of Internet Protocol (IP) packets with either a forged source address or using an unauthorized IP address. Such spoofing is now widely used to mask the source of cyberattacks over the Internet, both by criminal gangs as well as by governments.

It is difficult to predict the consequences of any new technology. Along with benefits there are often some downsides that later emerge. One such downside was the emergence of *spam* emails. Spam consists of unsolicited commercial emails that are now sent out to millions of email users in a bulk mailing. The email spam costs the spammer very little to send and even if only a tiny percentage of recipients respond it can be a very profitable business. One of the first spam emails was sent to the ARPANET community by

Fig. 12.1. An example of worldwide botnet detections by the Microsoft Digital Crimes Unit.



an overenthusiastic DEC marketing representative in 1978. Since then, the volume of email spam has grown enormously. A 2003 study estimated that more than half the email transmitted over the Internet was spam and that more than 90 percent of all spam email was sent by just 150 people. By 2011, according to one estimate, spam emails accounted for more than 80 percent of all email sent over the Internet. Increasingly, these spam emails are not sent by identifiable individuals but by *zombie computers* or *botnets* (Fig. 12.1) as we discuss in the following text. Botnets are made up of personal computers belonging to ordinary users whose machines have been taken over by computer malware that can be instructed to send out spam. Fortunately, *spam filters* are now available that can identify most spam emails and redirect them straight to the “junk” email folder.

Malware is short for *malicious software* and means software that is designed to gain unauthorized access to computers for a range of purposes, some relatively harmless and others definitely criminal. The popularity of the Unix operating system in universities and businesses in the 1970s and 1980s originally made Unix a prime target for *black hat* hackers, clever programmers who use their skills to gain unauthorized access to computer files. Nowadays, because of the success of the personal computer, Microsoft Windows is the operating system most under attack. In many cases, the hackers are able to gain control of the high-level system security privileges of the *system administrators*, the people responsible for keeping the computer system running. On the other side in this hacking war are the *white hat* hackers. These are ethical computer security experts who specialize in finding security loopholes and in defending computer systems from cyberattacks.

The techniques used by the black hats are many and varied. We begin by discussing a selection of the most common techniques before looking at the recent escalation in the use of malware for cyberwarfare. We then take a brief look at modern cryptographic systems that are designed to keep Internet communications secure from eavesdroppers and end with some comments on cookies, spyware, and privacy.

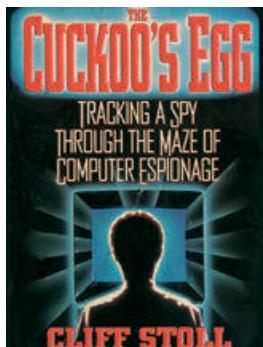


Fig. 12.2. A fascinating detective story about Cliff Stoll chasing a hacker during the ARPANET era.

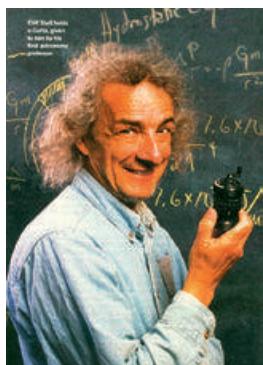
Cyberespionage

Clifford Stoll's (B.12.1) classic book *The Cuckoo's Egg* describes the complexity of tracking and prosecuting a black hat hacker (Fig. 12.2). Stoll was an astronomer turned system administrator for the computers at Lawrence Berkeley National Laboratory. The lab's computers ran Berkeley Unix and had two systems of accounting software for keeping track of the usage of these machines - one a standard Unix utility program and the other a homegrown program specific to Berkeley. From a seventy-five-cent discrepancy in the computer accounts at Lawrence Berkeley National Laboratory in 1986, Stoll deduced that someone was hacking into the lab's system. By sleeping in the lab and being alerted to every incoming computer connection, Stoll was able to record the exact keystrokes that had been used when the offense occurred. The results were surprising.

The hacker had gained access to one of Stoll's computers by guessing the password for an old, inactive user's account. When in the system, he then used a bug in the popular GNU-Emacs editor program to trick the computer into giving him the same privileges as a system administrator, so-called super-user or root privileges. This bug allowed him to move a file from his user area into what should have been an area of memory restricted to the system manager. The GNU software did not check whether the area was in the protected system software memory space. Once in this privileged area, the hacker then ran a counterfeit version of a standard Unix program, *atrun*, which runs queued up jobs at regular intervals. This unauthorized program is the cuckoo's egg of the title of the book - named for the cuckoo's trick of laying its eggs in nests of other birds. Running the counterfeit program allowed the hacker to gain the super-user privileges of a system administrator. He then restored the real Unix *atrun* program and erased his tracks from the system log so that the systems administrators would see nothing wrong. He also scanned all email messages for references to "hacker" and "security" and used his new privileges to kill the program of any user who he thought might have been monitoring his activity.

The situation was extremely serious: the hacker could read anyone's email, access or delete any file, and set up a new, hidden account that could provide him with a "backdoor" into the computer known only to him. All of the data stored on the computer was now at risk. Moreover, from his position as super-user, he was able to explore not only all the other computers at the Berkeley Lab connected by the Local Area Network (LAN), but also the computer systems connected to Berkeley through the ARPANET.

Stoll watched the hacker systematically attempting to break into several military computer installations by guessing passwords or by using unprotected guest or visitor accounts. It was surprising how many supposedly secure military sites still used the standard factory password settings for their super-user system administrator accounts. After a long chase - and remarkable initial indifference from the Federal Bureau of Investigation (FBI), the Central Intelligence Agency, and even the National Security Agency (NSA) - the trail led to West Germany (Fig. 12.3). The hacker, Markus Hess, was part of a group selling sensitive information obtained from these U.S. military computing systems to the Soviet Union.



B.12.1. Clifford Stoll is a U.S. astronomer and author who is probably best known for his book *The Cuckoo's Egg*. This tracked a hacker who had broken into Stoll's computer at Lawrence Berkeley Laboratory back to Hanover, Germany.



Fig. 12.3. The NSA was established in 1952 to handle secret communications and gather intelligence.

The Berkeley hacker used another technique to steal passwords: he had installed a *Trojan horse* program. In Virgil's *Aeneid*, when the Greeks pretended to abandon their siege of the city of Troy, they left behind a giant wooden horse. The citizens of Troy took the horse into the city and celebrated the defeat of the Greeks. In fact, the horse was full of Greek soldiers and the Trojans had brought the enemy inside their defenses, leading to the sacking of their city. A Trojan horse program does much the same thing for a computer system. It hides malicious or harmful code inside an apparently harmless program so that it can get control and do damage. At Berkeley, the hacker produced his own version of the standard login program to capture users' passwords. A would-be user was greeted by what looked like the normal login message:

**WELCOME TO THE LBL UNIX-4 COMPUTER
PLEASE LOGIN NOW**

Login:

After the user typed the account name, the system then asked for the password:

ENTER YOUR PASSWORD:

The user entered the password, which was copied along with the account name into a file set up by the hacker. The program then responded:

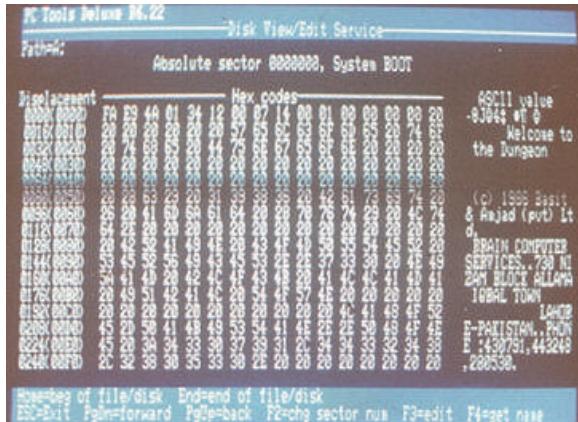
SORRY, TRY AGAIN

The user is then returned to the real login page and logs in as usual, unaware that the account details and password have been stolen. Such Trojan horse techniques are now widely used to capture private personal information and bank account details.

Viruses, rootkits, and worms

In principle, the damage caused by a Trojan horse program is restricted to one computer. A computer *virus*, as the name implies, is nastier in that it is designed to spread to other computers. The code for a virus is a small set of instructions incorporated into an application rather than a complete, stand-alone program. Initially, computer viruses were spread by the exchange of infected floppy disks but are now more typically spread using the Internet by getting users to click on harmless-looking email attachments like a photograph or a document. One of the first major virus attacks was the "Brain" virus (Fig. 12.4). Two Pakistani brothers created it in 1986, targeting bootable floppy disks for PCs running MS-DOS. A *bootable floppy disk* was one that held its own operating system and was usually used to restart a failed system or to install a new operating system. When the PC was *booted* (started up) from the infected disk, the computer loaded the Brain virus before executing the original MS-DOS code. The virus hid itself from the user by reporting the sectors of the floppy disk on which it was installed as damaged. If the user actually checked the boot code on the disk, the original uninfected code would be displayed rather than the modified code including the virus. In this case, the result was relatively harmless: the virus spread an advertisement for the brothers' company with its name and contact details, a genuine example of "viral" advertising.

Fig. 12.4. A screenshot of the BRAIN virus in 1984; it was one of the first PC viruses.



After the example of the Brain virus, hackers developed many thousands of new viruses, often using clever new techniques to help them spread. One of the most striking was produced in Germany in 1987. It was called the Cascade virus because it made the characters on the screen appear to fall to the bottom. This virus also introduced a new level of sophistication by using *encryption* techniques, which convert messages into secret code, to hide the details of its internal workings. We will discuss encryption later in this chapter. It was this explosion of computer viruses in the 1990s that led to the creation of a whole new industry – with antivirus companies now providing software to combat malware.

As a footnote, the term *computer virus* was probably first used by Len Adleman, a professor at the University of Southern California, well known for his contributions to cryptography. His student Fred Cohen was studying computer infections and defined a virus as “a computer program that can affect other computer programs by modifying them in such a way as to include a (possibly evolved) copy of itself.”² In November 1983, Cohen demonstrated a computer virus that infected the Unix file directory program. After some other experiments with program infections, Cohen examined the theoretical difficulty of detecting computer viruses. His PhD thesis in 1986 showed that there is no way of definitively detecting a virus. The best we can do is to assemble a collection of tricks and informal techniques, sometimes known as *heuristics*, to supplement our guesswork.

The Brain virus was one of the first to use *cloaking* techniques to hide the program from common system administrator and diagnostic utilities. In Unix, the traditional name for the most privileged account is *root*, and software designed to give a user root privileges is sometimes known as a *rootkit*. The term *rootkit* is now applied more generally to types of malware that use cloaking techniques to make themselves invisible to antivirus software and standard system tools. Rootkits came to prominence in 2005 when the Sony BMG music group installed overaggressive copy protection measures on twenty million music CDs. When the CD was used, it secretly installed software that actually modified the operating system to prevent CD copying. Moreover, the software was very difficult to remove and used the same rootkit cloaking techniques as conventional malware to hide its presence. The scandal came to light when security researcher Mark Russinovich (B.12.2) posted a detailed technical



B.12.2. Mark Russinovich was a security researcher at his Winternals company when he became a victim of Sony BMG's CD rootkit. His subsequent blog post on the technical aspects of the rootkit showed how it installed itself and modified the operating system of an unsuspecting user. Russinovich is now a Technical Fellow at Microsoft and the author of the novels *Zero Day* and *Trojan Horse*.



B.12.3. Robert Morris Sr. (1932–2011) was chief scientist of the NSA's National Computer Security Center at the time of Clifford Stoll's cuckoo's egg experiences with cyberespionage. Before he joined the NSA in 1986, Morris had been a researcher at Bell Labs working on both the Multics and Unix operating systems.

description of the Sony rootkit on his blog in October 2005. He also discovered that the software created new security loopholes and could lead to system crashes. Sony BMG's reaction to this revelation was initially: "Most people don't even know what a rootkit is so why should they care about it?"³ However, the company eventually recalled and replaced the affected CDs and abandoned its extended copyright protection software. Mikko Hypponen, chief research officer at the Finnish-based security company F-Secure, commented:

[The] Sony rootkit was one of the seminal moments in malware history. Not only did it bring rootkits into public knowledge, it also gave a good lesson to media companies on how not to do their DRM [digital rights management] solutions.⁴

The term *computer worm* is generally used to describe malware that is designed to spread from computer to computer but, unlike a virus, which must attach itself to a program or file to spread, a worm is a complete program capable of replicating all by itself. At Xerox PARC in 1978, John Shoch was experimenting with a program that could seek out Alto machines on the Ethernet that were not being used, boot up the machine to do some work, and replicate by sending copies of itself to other idle machines on the network. One of his experiments went wrong and, after leaving his program running overnight, Shoch was awakened by angry users complaining that he had crashed their Altos. Eradicating the worm proved very difficult, and it was fortunate that he had equipped his worm program with a "suicide capsule" that he was able to activate. Shoch called his program a *worm*, after the idea of the "Tapeworm," software that runs by itself in John Brunner's science fiction novel *The Shockwave Rider*.

Worms came into public prominence through the "Internet worm" attack on the ARPANET in 1988. Clifford Stoll, then at Harvard, described this "Internet worm" attack in graphic detail:

As fast as I'd kill one program, another would take its place. I stomped them all out at once: not a minute later, one reappeared. Within three minutes there were a dozen.⁵



B.12.4. Robert Morris Jr. was a graduate student at Cornell when he created the first worm on the ARPANET in 1988. He was the first person to be convicted under the USA Computer Fraud and Abuse Act. He is now a tenured professor at MIT.

Stoll informed Bob Morris (**B.12.3**), chief scientist at the NSA, whom he knew from his investigation of the Berkeley hacker, of the ongoing worm attack. Stoll was not amused to be called back a few hours later by someone from the NSA who asked if he was the person who had written the worm program! While other ARPANET node system administrators across the United States were decrypting the worm program, Stoll tracked down the place where the worm had been released. By a supreme irony, the trail led back to Bob Morris Jr. (**B.12.4**), a graduate student at Cornell University and the son of Bob Morris Sr. of the NSA. The Morris worm was not the first worm program, but it was certainly one of the most damaging. Stoll estimated that it infected two thousand machines within fifteen hours.

Morris's worm was a significant escalation in malware for two reasons. First of all, the program automated all sorts of tricks that a hacker might use in attempting to break into a computer system. Given access to one computer, the worm would first check if it was automatically given privileges to run programs



Fig. 12.5. The infamous Morris worm was only a short C program, yet it shut down large portions of the ARPANET in November 1988.

on other computers; then it would try a long list of common passwords. If these attempts failed, it would then try some other vulnerability, such as a flaw in the Unix Sendmail program, well known to computer experts at the NSA. The second reason for its importance was that if all these attempts failed, Morris had exploited a new type of bug called *buffer overflow*. The Unix operating system is written in the C programming language, and the first book about C was written by Bell Labs researchers Brian Kernighan and Dennis Ritchie. The book shows how to write a program to read a series of input characters into computer memory using an area of memory called a *buffer*. In their example code, the size of the buffer was specified but not whether the number of characters being entered actually exceeded this size. The younger Morris realized that the extra characters would overwrite the rest of the program's data and instructions. By placing judicious machine instructions in these overflow characters, a hacker could use this flaw to gain the root privileges of a super-user. Morris also encrypted the virus software to make it more difficult to find out what the program did and also used several techniques to avoid detection. The worm infected thousands of computers, and system managers took several days to disinfect their computers. Morris was convicted of a felony in May 1990 and sentenced to three years of probation, four hundred hours of community service, and a \$10,000 fine (Fig. 12.5).

The story had a happy ending for Morris. After his conviction, Xerox PARC invited him to become an intern student there and he is now a professor at MIT. However, an unfortunate outcome of Morris's worm was that it demonstrated a new way of attacking computers. Such unchecked memory buffers occurred in almost all Unix programs and also in Windows. After a hacker called "Aleph One" put up a detailed "instruction manual" (Fig. 12.6) on the Web in 1996, buffer overflow became a relatively straightforward technique for black hats to adapt. In 1992, there were estimated to be around 1,300 viruses or worms; in 1996, more than 10,000; and by 2002, more than 70,000. By 2003, the Slammer worm had set a record for spreading faster than any previous malware, infecting seventy-five thousand computers in just ten minutes.

Fig. 12.6. Aleph One's paper on the buffer overflow vulnerability.

.oO Phrack 49 Oo.
Volume Seven, Issue Forty-Nine
File 14 of 16
BugTraq, r00t, and Underground.Org
bring you

XXXXXXXXXXXXXXXXXXXXXXXXXXXX
Smashing The Stack For Fun And Profit
XXXXXXXXXXXXXXXXXXXXXXXXXXXX

by Aleph One
aleph1@underground.org

'smash the stack' [C programming] n. On many C implementations it is possible to corrupt the execution stack by writing past the end of an array declared auto in a routine. Code that does this is said to smash the stack, and can cause return from the routine to jump to a random address. This can produce some of the most insidious data-dependent bugs known to mankind. Variants include trash the stack, scribble the stack, mangle the stack; the term mung the stack is not used, as this is never done intentionally. See spam; see also alias bug, fandango on core, memory leak, precedence lossage, overrun screw.

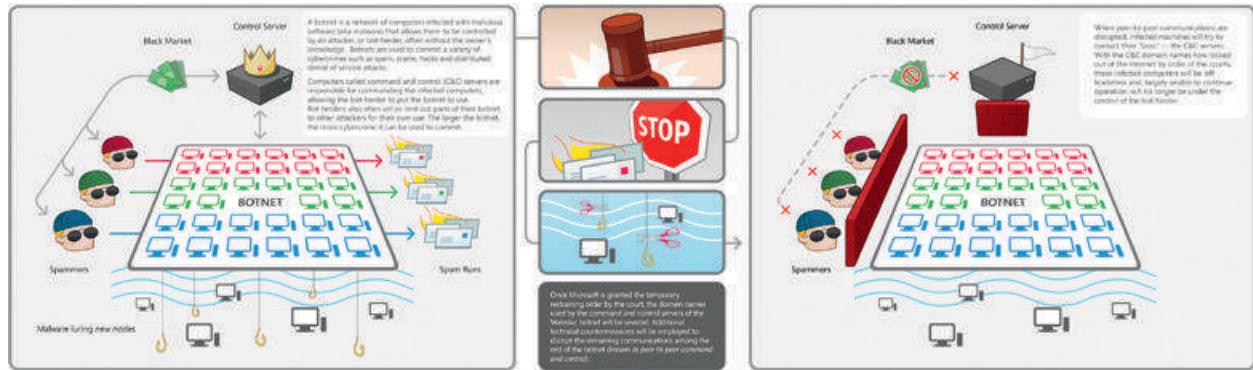


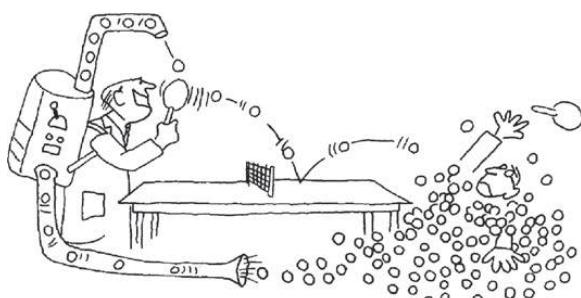
Fig. 12.7. Nefarious botnet programs work by hijacking millions of computers, usually without their owners' knowledge.

Botnets and zombie computers

The last decade has seen a dramatic rise of hacking for profit by criminal organizations. *Botnets* are collections of computers that have been taken over by techniques such as those described above and are controlled by so-called *bot-herders* (see Fig. 12.7). *Bot* is short for *robot program*, and sometimes these enslaved computers are known as *zombie computers*. The botnets can be used to conduct *denial of service attacks* on specific websites (see Fig. 12.8) – such attacks try to shut down a site by bombarding it with so many requests that the system is forced to shut down, denying service to legitimate users. Botnets can also be used to send spam or to capture personal details by *key logging* – capturing a user's keystrokes. A recent example is the Conficker botnet that first appeared in 2008. It was estimated to have infected more than ten million computers around the world and to have the capacity to send an incredible ten billion spam emails per day. Mark Bowden's book, *Worm*, details how the white hat security community collaborated with Microsoft to contain and partially eliminate the threat to the Internet posed by Conficker. However, a 2012 Microsoft report states that

... the Conficker worm was detected approximately 220 million times worldwide in the past two and a half years, making it one of the biggest ongoing threats to enterprises. The study also revealed the worm continues to spread because of weak or stolen passwords and vulnerabilities for which a security update exists.⁶

Fig. 12.8. This cartoon with the original caption "... filibustering destroys communication" captures the essence of a "denial of service" attack.



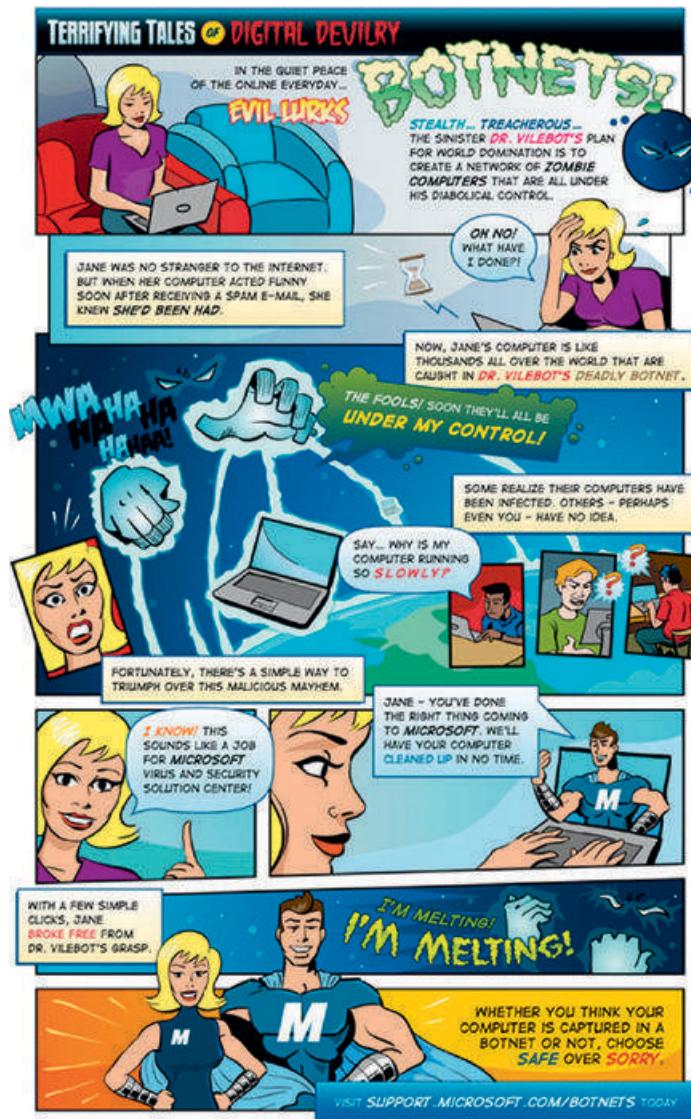
In another recent example, the Microsoft Digital Crimes Unit worked with the financial industry and the FBI “to disrupt more than 1,400 Citadel botnets which are responsible for over half a billion dollars in losses to people and businesses.”⁷

A last example is particularly worrying. In the case of the Nitol botnet (Fig. 12.9), Microsoft found that nearly 20 percent of brand new PCs purchased through unsecure Chinese supply chains were already preinfected with Nitol malware.

A supply chain between a manufacturer and a consumer becomes unsecure when a distributor or reseller receives or sells products from unknown or unauthorized sources. In Operation b70, we discovered that retailers were selling computers loaded with counterfeit versions of Windows software embedded with harmful malware.⁸

This malware is particularly worrisome since it can be spread to friends and colleagues through a USB memory stick.

Fig. 12.9. A cartoon strip from Microsoft showing how to evade evil botnets.



Cyberwarfare

The latest escalation in malware is the potential to use worms for *cyber-warfare*, politically motivated hacking for purposes of spying or sabotage. It is now believed that the Stuxnet worm discovered in the summer of 2010 was engineered by U.S. and Israeli computer experts specifically to attack centrifuges at the Iranian uranium fuel-enrichment plant in Natanz, a site suspected of being a center for building a uranium-based atomic bomb. A series of high-speed centrifuges is needed to separate the rare, bomb-grade uranium-235 isotope from the much more common uranium-238 isotope present in uranium ore. An industrial control system manufactured by Siemens AG manages the centrifuges at the Natanz plant. This system uses a special-purpose computer called a *programmable logic controller* (PLC) that is programmed using Siemens software called Step-7. To spread itself throughout the plant, the Stuxnet worm exploited several previously unknown bugs – known as *zero day* bugs – in the Microsoft Windows XP operating system. In this way, the worm seized control of the PCs and substituted its own version of Siemens's Step-7 PLC code. This code modified the operation of the centrifuges yet reported to the operator that everything was fine. Thus the Step-7 malware used rootkit techniques to conceal its presence. The writers of worm code needed a very deep knowledge of both Windows and Siemens industrial control systems as well as detailed information about the centrifuge installation at the Natanz fuel-enrichment plant. It is likely that the worm was introduced into the Natanz using a USB memory stick since the plant is believed to be *air-gapped* – not connected to the Internet. A recent book, *Confront and Conceal*, by New York Times reporter David Sanger, describes operation *Olympic Games*, the codename for Stuxnet development and deployment, and details how the worm escaped to the Internet.

How much damage did Stuxnet cause? One report suggests that as many as a thousand centrifuges at Natanz or around 10 percent of the total needed to be replaced. In the long run, what may be of more significance than the cyberattack on Natanz is that the Stuxnet worm represents a blueprint for the construction of malware capable of attacking a wide range of industrial control systems, which form a key part of the modern world's critical infrastructure.



Fig. 12.10. The word *cryptos* (κρυπτός) in Greek means hidden. This is the Kryptos sculpture located in front of CIA headquarter in Langley. There are four messages on the sculpture; three of them have been deciphered, the fourth is so far unbroken.

Cryptography and the key distribution problem

The science of cryptography dates back to ancient times (Fig. 12.10). It consists of techniques for *encoding* the information in a message – that is, for putting the information into a form that can only be read or *decoded* by the intended recipient. According to the Roman historian Suetonius, Julius Caesar used a method called a *shift cipher* to encode secret government messages:

If he had anything confidential to say, he wrote it in cipher, that is, by so changing the order of the letters of the alphabet, that not a word could be made out. If anyone wishes to decipher these, and get at their meaning, he must substitute the fourth letter of the alphabet, namely D, for A, and so with the others.⁹



Fig. 12.11. Cipher disk invented by the Renaissance artist Leon Battista Alberti (1404–72). The brass inner disk can be rotated to align its letters with the letters of the outer circle.

A shift cipher consists of a key or number, known only to sender and receiver, which tells you how far to shift a second alphabet that is written under the first one (Fig. 12.11). Alan Turing, with help from Polish intelligence and others at Bletchley Park in the United Kingdom, built one of the first primitive computers to break the German Navy's Enigma codes. To break the German High Command's more complex Lorenz codes, Tommy Flowers built Colossus, arguably the first serious digital computer. With the advent of modern computers, cryptographers no longer needed to rely on a mechanical cipher machine to do the encryption and decryption. A computer can do the work of a complex cipher machine and still operate many times faster than any mechanical device. Since computers operate on binary numbers, messages must first be converted into a series of 1s and 0s according to some convention. There are now standard ways to encode characters and words into binary numbers. Once the message has been converted into a string of bits, encryption proceeds by scrambling the bits according to a method specified by a key that is shared by sender and receiver.

There are two main classes of cryptosystems, which are distinguished by whether the encryption key is shared in secret or in public. Gilbert Vernam (B.12.5) of AT&T proposed the one-time secret-key system in 1918. It is the only cryptosystem that provides absolute security. However, the system requires a key that is as long as the message and the keys must never be reused to send another message. Spies received a fresh set of keys in the form of a tear-off pad. After sending a message, the sender tore off the sheet with the used key and destroyed it. For this reason, the system is sometimes known as a *one-time pad*. When the Bolivian army captured the Marxist revolutionary Che Guevara in 1967, they found he had a list of random numbers that allowed him to send secret messages to Fidel Castro in Cuba. Guevara could do this securely over any radio link because he and Castro were using Vernam's one-time pad system.

In cryptology, the three participants in any discussion of coded messages are traditionally Alice, Bob, and Eve. Alice is the sender who wants to encrypt a message and send it securely to Bob. Bob is the receiver, who gets the message and wants to decrypt it and discover its meaning. Eve is a potential eavesdropper who wants to listen in and break the code. The one-time pad is secure because Alice encrypts the message using a random number as long as the message. Bob has the same key and can easily decrypt the message. The particular random number is only used once. Although this system is perfectly secure in principle, its weakness in practice lies in the fact that Alice and Bob have to share the same key and, since the keys are used only once, they need a great deal of them. The keys have to be distributed to Alice and Bob using some secure method, such as delivery by courier or a personal meeting. During World War II, the Russians foolishly reissued some one-time pads. This carelessness allowed U.S. cryptanalysts to decrypt a large number of previously undecipherable messages that they had intercepted over the years. This large-scale decoding effort was code-named the Venona project (Fig. 12.12). It was transcripts from this project that identified the atomic spy code-named CHARLES as the Los Alamos physicist Klaus Fuchs.

The weakness of secret-key encryption is the problem of distributing the keys safely. During World War II, the German military had to distribute



B.12.5. Gilbert Vernam (1890–1960) came up with the idea of unbreakable encryption using so-called *one-time pads* of secret code numbers. The system was used by Che Guevara to communicate with Fidel Castro in Latin America.

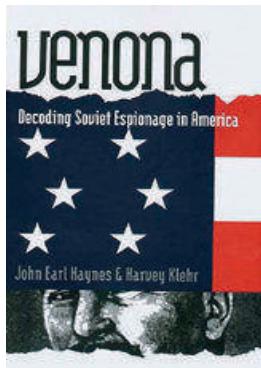


Fig. 12.12. The Venona project was a U.S. counterintelligence program to decrypt messages sent by the Soviet Union's intelligence agencies. The secret program was operational for more than forty years. Its existence was only revealed in 1995 after the end of the Cold War. The program identified Klaus Fuchs as the Manhattan Project spy who gave the plans for the atomic bomb to Stalin.

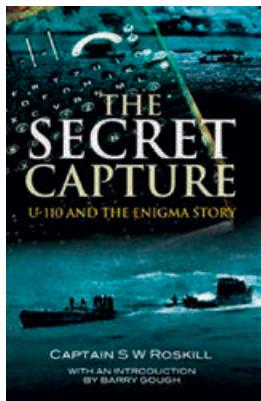


Fig. 12.13. *The Secret Capture* tells the story of how the British destroyer HMS Bulldog captured the German submarine U-110. The British sailors were able to retrieve the codebooks and an Enigma machine from the submarine and these were sent to the code breakers at Bletchley Park.

books containing a month's supply of keys to each operator of the Enigma code machine. For the U-boat fleet operating in the North Atlantic Ocean, this was a major logistical challenge and also a critical vulnerability. Ian Fleming, creator of James Bond, was a member of the United Kingdom's Naval Intelligence Division during the war. He suggested a James Bond-style plan called "Operation Ruthless" to capture the Enigma codebooks from a German ship. Although this particular operation was never carried out, the Allies did manage to capture intact Enigma codebooks from German weather ships and U-boats, enabling them to learn the locations of the Atlantic U-boat packs (Fig. 12.13).

The United States adopted the Data Encryption Standard (DES), a standard method of coding messages, in 1976. The DES was based on a system devised by the German-born cryptographer Horst Feistel, working at IBM's Thomas J. Watson Research Center in New York. It is widely believed that the U.S. government only allowed 56-bit keys so that the DES system was safe enough for normal users but not impossible for the NSA to break. Banks who needed to send secure messages of detailed transactions to each other were major users of encryption. To solve the problem of key distribution, banks employed dispatch riders who had to be thoroughly investigated and then equipped with padlocked briefcases. The costs of maintaining such a system rapidly became a major expense.

Diffie-Hellman key exchange and one-way functions

The way out of all these problems was to find a way for Alice and Bob to agree on a secret key without ever having to meet, in spite of Eve trying to listen in and discover the key. Remarkably, in 1976, agreeing on a secret key without meeting was shown to be possible. In his wonderful account of ciphers and cryptography, *The Code Book*, Simon Singh says of this new method of exchanging keys, "It is one of the most counterintuitive discoveries in the history of science"¹⁰ and adds, "This breakthrough is considered to be the greatest cryptographic achievement since the invention of the monoalphabetic cipher, over two thousand years ago."¹¹

The system that allows Alice and Bob to establish a secret key through a public discussion is called the *Diffie-Hellman key exchange*, after the inventors Whitfield Diffie and Martin Hellman (B.12.6). Hellman was a professor at Stanford University, and Diffie enrolled as his graduate student so they could both study the key distribution problem. Diffie and Hellman had realized that the solution to the problem required the use of a mathematical relationship called a *one-way function*. A two-way mathematical function is reversible in that it is easy to undo; a one-way function, as the name implies, is easy to do but very difficult to undo. Singh gives the following analogy: "Mixing yellow and blue paint to make green paint is a one-way function because it is easy to mix the paint but impossible to unmix it."¹² We can use this paint-mixing analogy to explain how Alice and Bob can establish a secret key without Eve finding out, even though she is able to monitor their public exchanges. We assume that each of the participants has a pot of yellow paint, and Alice and Bob each have another pot with their own secret color. They proceed as follows:

If Alice and Bob want to agree on a secret key, each of them adds one liter of their own secret color to their own pot of yellow paint. Alice might add a peculiar shade of purple, while Bob might add crimson. Each sends their own mixed pot to the other and we assume that Eve can see and even sample these mixtures as they are sent between Alice and Bob. Finally, Alice takes Bob's mixture and adds one liter of her own secret color, and Bob takes Alice's mixture and adds one liter of his own secret color. Both pots should now be the same color, because they both contain one liter of yellow, one liter of purple and one liter of crimson. It is the exact color of the doubly contaminated pots that is used as the key.

Does Eve know the secret key? No, she doesn't. She saw (and possibly sampled) the two partial mixtures that passed by her: "yellow and purple" and "yellow and crimson." If Eve combines these mixtures – the only operation she could do on her own – she will only end up with a mixture containing "yellow and yellow and purple and crimson." In order to find the secret key she would need to remove or "unmix" one unit of yellow from this mixture. Since she cannot unmix one unit of yellow she cannot generate the same color as Alice and Bob and thus does not know the key.¹³

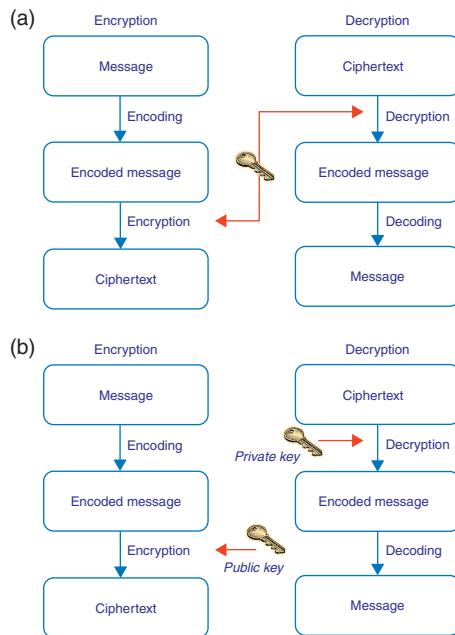
So although Eve can intercept the pots of paint being exchanged, she cannot work out Alice's and Bob's secret keys because mixing paint is a one-way function.

The actual mathematical one-way function used in the Diffie and Hellman key exchange proposal was based on *modular arithmetic*. Calculations in modular arithmetic are done with a count that resets itself to zero every time a certain number, known as the *modulus*, has been reached. Modular arithmetic is like telling time using the numbers on a clock face. For example, $9 + 7$ in normal arithmetic equals 16. However, in modular arithmetic with a modulus of 12 ("mod 12" arithmetic, also called *clock arithmetic*), the result of $9 + 7$ is 4. If it is 9 o'clock in the morning then seven hours later it will be 4 o'clock in the afternoon. Because the hour number starts over after it reaches 12, the modulus is 12. In normal arithmetic, the result of adding two numbers grows as the numbers being added are larger. With modular arithmetic, the numbers can grow just to the value of the modulus. Although this key exchange system was a great breakthrough in cryptography, it still required that Alice and Bob exchange several messages to establish the shared secret key. The Diffie-Hellman key exchange protocol was also fundamentally a two-party protocol rather than a broadcast protocol that allowed Alice or Bob to communicate securely with



B.12.6. Whitfield Diffie and Martin Hellman are the inventors of the Diffie-Hellman key exchange protocol. This is remarkable process by which Alice and Bob can agree on a secret key using an open link that is vulnerable to access by an eavesdropper, Eve.

Fig. 12.14. (a) Symmetric encryption shares the same encryption key between sender and receiver. (b) Asymmetric encryption uses a different encryption key at each end of the communication.



others. Another breakthrough was needed to arrive at a secure and convenient cryptographic method that eliminated key exchange bottlenecks.

Up until 1975, all the encryption techniques in history had been *symmetric*, meaning that the key to unscramble the message was the same as the key used to scramble it in the first place (Fig. 12.14a). In the summer of 1975, Diffie outlined the idea for a new type of cipher that used an *asymmetric key pair*, one in which the encryption key and the decryption key were different but mathematically related (Fig. 12.14b). Although he showed that such a system could work in theory, Diffie was unable to find a suitable one-way function to actually carry out his idea. If such a system could be found, then it could work as follows. Alice would have two keys, one for encryption and one for decryption. She can make her encryption key public, her “public key,” so that everyone has access to it, but she keeps her decryption key secret as her “private key.” Now if Bob wants to send a message to Alice, he can encrypt his message using Alice’s public key. When she receives the message, Alice is able to decrypt the message using her private key, secure in the knowledge that Eve, who only knows Alice’s public key, would be unable to make sense of the message. This is the essence of the cryptographic system called *public-key cryptography*.

RSA encryption and pretty good privacy

The race to make asymmetric ciphers a reality was won by three researchers working in the Laboratory for Computer Science at MIT: Ron Rivest, Adi Shamir, and Len Adleman (B.12.7). Their resulting scheme is now known as *RSA encryption*, and it depends on modular exponentiation and the difficulty of factoring large numbers. The scheme relies on the fact that multiplication of two large prime numbers, p and q , to get the number N is very easy and

RSA-129			
3490529610	3276813299	1143816257578888766	
8476509491	3266709549	92357799761466120104	
4784651990	9619881908	18296721242362582561	
3898133417	X 3446141337	* 84293570693524573385	
7646384933	7642987992	7830597123563958705C	
8784399082	9425397982	589890754759929002E	
0577	88533	879543541	

Fig. 12.15. The number RSA-129. Multiplication is computationally “very easy” whereas factorization of a number into its constituent prime numbers is computationally “very hard.” This is the basis for the security of the RSA Public-Key Cryptographic system.

9686961375
5882905759991
1629822514570
43552093081389
140922543512093
517669314176628
9962801339486
3989962801339486
1222631232263123
57151571515715157

Fig. 12.16. The original encrypted text of Gardner’s challenge.

quick to do with a computer, but the reverse problem of factoring N – deducing the prime numbers that when multiplied together produce N – is very difficult. Martin Gardner, in his “Mathematical Games” column in *Scientific American*, explained public-key cryptography and the RSA asymmetric cipher in August 1977. He issued a challenge to his readers by giving them a ciphertext to decode that had been encrypted using a public key N, which he published. The public key was a 129-digit number known as RSA-129. To decrypt the message, the readers had to *factor* (break up) this number into its two *prime factors*, the prime numbers that were multiplied together to produce the 129-digit number. It was almost seventeen years before a team of six hundred volunteers assembled sufficient computing power to discover the two prime number factors (see Fig. 12.15). When at last deciphered, Gardner’s message read, “The magic words are squeamish ossifrage” (Fig. 12.16). Nowadays, given the huge increase in computing power since 1977 generated by Moore’s law, much larger values than RSA-129 need to be used to secure messages and information. These numbers are so large that it is estimated it would take all the computing resources on the planet many thousands of years to factorize such large numbers. However, as we will see later, such public-key systems could be vulnerable to attack by a quantum computer if such a computer could be built.

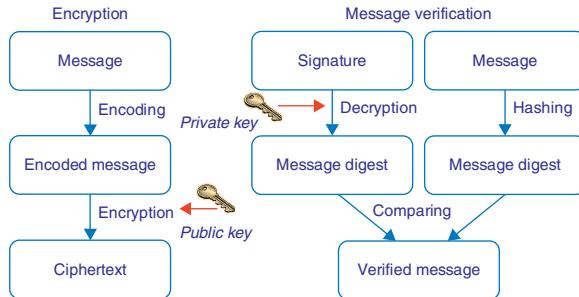
In the 1980s, only governments, the military, and large businesses had computers that were powerful enough to use RSA efficiently. Phil Zimmermann (B.12.8), a software engineer specializing in cryptography and data security, believed that everyone should have the same guarantee of privacy in communications made possible by RSA encryption. Such a capability is particularly important for human rights activists operating in countries with repressive regimes. Even in more open countries, privacy of communications can be regarded as a basic democratic freedom. The problem is that this same freedom would also severely limit the ability of governments to monitor communications between criminals.

Zimmermann wrote a program that he called *Pretty Good Privacy* (PGP), a name inspired by Ralph’s *Pretty Good Grocery*, a business in Garrison Keillor’s fictional town of Lake Wobegon. In his PGP program, Zimmermann implemented a fast version of the RSA public-key system. Unfortunately, he also chose to ignore the fact that the RSA technology was patented. Zimmerman apparently



B.12.7. Ron Rivest, Adi Shamir, and Len Adleman are the inventors of RSA public-key cryptography protocol, which is now in widespread use. In their scheme, Alice now has two keys – an encryption key that she makes public and her private decryption key.

Fig. 12.17. The mechanism of a digital signature, an electronic signature that can be used to authenticate the identity of the sender of a message or the signer of a document.



hoped that the patent owner, Public Key Partners, would give him a free license since PGP was intended for use by individuals and not for commercial use. It was left to a group of cryptography researchers at MIT to make PGP legal by removing Zimmerman's implementation of the RSA algorithm and replacing it with a legal version with an appropriate RSA license.

The PGP software also incorporated *digital signature* authentication. Digital signature technology addresses the problem that, without a handwritten signature, it is difficult to be sure who actually sent an email message. Bob can use Alice's public key to send an encrypted message to her, but so can Eve, masquerading as Bob. So how can Alice check that the message is really from Bob? One way of verifying that the message was indeed sent by Bob goes as follows. Bob first encrypts the message using his private key and then does a second encryption, encrypting the resulting message using Alice's public key. When Alice receives the message, she begins by decrypting it by first using her private key and then uses Bob's public key to decrypt the still encrypted message. This way she can verify that the message came from Bob (Fig. 12.17).

In 1991, Zimmerman became worried that the U.S. Senate would pass a bill that would outlaw the use of such encryption technology, so he arranged for his PGP code to be posted on an Internet bulletin board. In response to this, the U.S. government, concerned about its ability to decipher communications between criminals or terrorists, accused Zimmerman of illegally exporting weapons technology. After some difficult years for Zimmerman, the government eventually dropped the case. Meanwhile, the code for the legal version of PGP was published in a book from MIT Press and could be legally exported from the United States. Ron Rivest summarized the basic argument against prosecuting Zimmerman as follows:

It is poor policy to clamp down indiscriminately on a technology just because some criminals might be able to use it to their advantage. For example, any U.S. citizen can freely buy a pair of gloves, even though a burglar might use them to ransack a house without leaving fingerprints. Cryptography is a data-protection technology, just as gloves are a hand-protection technology. Cryptography protects data from hackers, corporate spies, and con artists, whereas gloves protect hands from cuts, scrapes, heat, cold, and infection. The former can frustrate FBI wire-tapping, and the latter can thwart FBI fingerprint analysis. Cryptography and gloves



B.12.8. Phil Zimmermann is the creator of PGP, an email encryption software package. Originally designed as a human rights tool, PGP was published for free on the Internet in 1991. This made Zimmermann the target of a three-year criminal investigation by the U.S. government, which held that export restrictions for cryptographic software were violated when PGP spread worldwide.

are both dirt-cheap and widely available. In fact, you can download good cryptographic software from the Internet for less than the price of a good pair of gloves.¹⁴

Twenty years after the publication of Zimmermann's PGP software, strong encryption technology is now widely available, and governments and police forces round the world have had to adapt to the new reality.

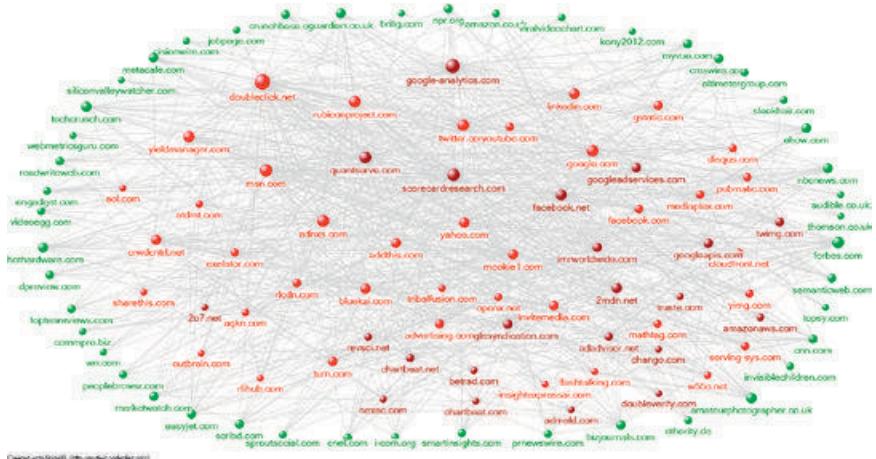
Although encryption using PGP software provides a very high level of security, it proved too complex for the average Web user. Netscape introduced a procedure called the *secure sockets layer* (SSL) to protect e-commerce transactions over the Internet. Without intervention from the user, the browser and the web server use the SSL protocol to automatically exchange public keys and to agree on a third, secret *session key* to encrypt the information being transmitted only for the current session. Instead of using the http protocol, the link to the website now uses https (standing for HyperText Transfer Protocol Secure), which just applies the http protocol on top of a protocol called the Transport Layer Security (TLS) protocol, the successor to the SSL protocol. All the user sees is a padlock icon in the browser window. Clicking on the padlock gives the user a security report, which says, "This connection to the server is encrypted." The report also gives details of a *digital certificate*, a credential that certifies the identity of the remote computer. The certificate verifies that the public key belongs to the specific organization or owner of the website. An organization called a *certificate authority* (CA) issues digital certificates. The CA is what is called a "trusted third party" – that is, an organization trusted by both the subject of the certificate and by the user wishing to access that site. The result of all these measures is that users now have a secure channel by which they can communicate personal details such as credit card numbers or their Social Security number.

Cookies, spyware, and privacy

Web cookies were first used in communications over the Internet by Lou Montulli, a programmer at Netscape Communications in 1994. The company was developing e-commerce applications and wanted to find a way to keep a memory of a user's transactions so that it would be easy to implement a virtual shopping cart. A web cookie, also known as an *http cookie*, is a small amount of data that is sent from the website a user is visiting and stored in the browser on the user's computer. They were designed to provide a way for websites to remember the user's browsing activity. Cookies were first introduced into Netscape's browser in 1994 and into Microsoft Internet Explorer in 1995. Although the cookies were stored on the user's computer, users were not initially notified of their presence. Cookies are convenient in that they can be used to store passwords and credit card details. When a user revisits a website, the website can recognize the user through the information stored in the cookie.

The real threat to privacy, however, came with the introduction of *third-party tracking cookies* (Fig. 12.18). First-party cookies are associated with the IP address shown in the address bar of the user's browser. Third-party cookies are cookies that are downloaded from a different domain than that shown in the browser. These come about as follows. When a user downloads a web

Fig. 12.18. Third-party cookies allow tracking companies to track the browsing behavior of web users. The green circles are the websites visited by a user and the purple circles are the companies analyzing the user's behavior and selling the information to the red sites that serve targeted advertisements to the user.



page this may contain an advertisement linking back to a different website. This site sets a cookie that tells the ad broker service that the user clicked on this web page. When the user visits another website the same thing happens and another cookie is downloaded. In this way, an ad broker can build up a complete picture of the user's browsing history. This information can then be sold to advertising agencies that can generate targeted, personalized ads, specific to the interests of the user, as revealed by their browsing history.

Spyware is software that can hide itself on a computer and gather and transmit information back to a black hat without the owner's knowledge. Spyware is different from viruses or worms in that the software does not try to replicate itself or spread to other computers. The Trojan horse software used by the Berkeley hacker is a form of spyware, for collecting user login information. Spyware can also collect other types of data such as bank and credit card information. In addition, spyware can track the user's Internet activity and serve annoying pop-up ads or change the computer's security settings and disable antivirus software. Cookies are a form of spyware and antispyware software now usually reports the presence of third-party cookies and offers ways to remove them.

Cookies have serious implications for the privacy of Internet users. In 2000, the U.S. government established strict rules for setting cookies, and modern browsers now offer users the option to block all cookies. In its *Directive on Privacy and Electronic Communications* in 2002, the European Union introduced a policy requiring a user's consent for setting cookies. It stipulated that storing data on a user's computer can only be done if the user is provided with information about how this data will be used. This was later relaxed to exempt first-party cookies – as in virtual shopping carts – from this requirement of obtaining prior user consent.

Key concepts

- Buffer overflow
- Trojans, viruses, and worms

- Rootkits and botnets
- Cyberespionage and cyberwarfare
- Cryptography and key exchange
- One-way functions and RSA encryption
- Cookies and spyware



Etymology of “spam”

The name *spam* derives from a famous 1970 Monty Python sketch about a man and his wife ordering food in a café, where they are offered various menu items all based on SPAM, a trade-named canned meat product:

Man: Well, what've you got?

Waitress: Well, there's egg and bacon; egg sausage and bacon; egg and spam; egg bacon and spam; egg bacon sausage and spam; spam bacon sausage and spam; spam egg spam spam bacon and spam; spam sausage spam bacon spam tomato and spam ...

Vikings [starting to chant]: Spam spam spam ...

Waitress: ... spam spam spam egg and spam; spam spam spam spam spam baked beans spam spam spam ...

Vikings [singing]: Spam! Lovely spam! Lovely spam!

Waitress: ... or Lobster Thermidor a Crevette with a mornay sauce served in a Provençale manner with shallots and aubergines garnished with truffle paté, brandy and with a fried egg on top and spam.

Wife: Have you got anything without spam?

Waitress: Well, there's spam egg sausage and spam, that's not got much spam in it.

Wife: I don't want ANY spam!¹⁵

God rewards fools

As Whitfield Diffie and Martin Hellman pursued the key distribution problem, they were joined by graduate student Ralph Merkle, who shared their enthusiasm for solving what seemed to be an impossible problem. Hellman commented:

Ralph, like us, was willing to be a fool. And the way to get to the top of the heap in terms of developing original research is to be a fool, because only fools keep trying. You have idea number 1, you get excited, and it flops. Then you have idea number 2, you get excited, and it flops. Then you have idea number 99, you get excited, and it flops. Only a fool would be excited by the 100th idea, but it might take 100 ideas before one really pays off. Unless you're foolish enough to be continually excited, you won't have the motivation, you won't have the energy to carry it through. God rewards fools.¹⁶

A truly cryptic development

An interesting postscript to the encryption story takes us back to the secrecy that surrounded the cryptographic work on the Enigma and Lorenz codes at Bletchley Park during World War II. After the war, the government of the United Kingdom concentrated its code-breaking efforts in a new agency called the Government Communications Headquarters (GCHQ) in Cheltenham (Fig. 12.19). By the 1960s, the British military had recognized the need for secure communications between troops in the field but was concerned about the logistics and costs of key distribution. GCHQ set some of its researchers on the problem, and the result was that, as Simon Singh says: "By 1975, James Ellis, Clifford Cocks and Malcolm Williamson had discovered all the fundamental aspects of public-key cryptography, yet they all had to remain silent."¹⁷ It was not until 1997 that Cocks was finally allowed to present a brief history of GCHQ's independent discovery of public-key cryptography. The same zeal for secrecy of successive U.K. governments denied Tommy Flowers meaningful recognition for his pioneering work in building the Colossus computers after the end of World War II.



Fig. 12.19. An aerial photo shows the GCHQ, the British agency responsible for communications security, based in Cheltenham, U.K. Two Colossus computers from Bletchley Park went to GCHQ after the war; the remaining eight were destroyed on Churchill's orders.

I3 Artificial intelligence and neural networks

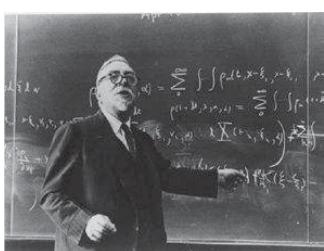
It is not my aim to shock you – if indeed that were possible in an age of nuclear fission and prospective interplanetary travel. But the simplest way I can summarize the situation is to say that there are now in the world machines that think, that learn and that create. Moreover, their ability to do these things is going to increase rapidly until – in a visible future – the range of problems they can handle will be coextensive with the range to which the human mind has been applied.

Herbert Simon and Allen Newell¹

Cybernetics and the Turing Test

One of the major figures at MIT before World War II was the mathematician Norbert Wiener (B.13.1). In 1918, Wiener had worked at the U.S. Army's Aberdeen Proving Ground, where the army tested weapons. Wiener calculated artillery trajectories by hand, the same problem that led to the construction of the ENIAC nearly thirty years later. After World War II, Wiener used to hold a series of “supper seminars” at MIT, where scientists and engineers from a variety of fields would gather to eat dinner and discuss scientific questions. J. C. R. Licklider usually attended. At some of these seminars, Wiener put forward his vision of the future, arguing that the technologies of the twentieth century could respond to their environment and modify their actions:

The machines of which we are now speaking are not the dream of the sensationalist nor the hope of some future time. They already exist as thermostats, automatic gyrocompass ship-steering systems, self-propelled missiles – especially such as seek their target – anti-aircraft fire-control systems, automatically controlled oil-cracking stills, ultra-rapid computing machines, and the like....²



B.13.1. Norbert Wiener's (1894–1964) name is mainly associated with the term *cybernetics*. Cybernetics is an interdisciplinary theory describing how complex systems regulate themselves using feedback mechanisms. Wiener was only eighteen when he earned his PhD degree in mathematics from Harvard University. During World War II, he worked on automatic control of antiaircraft guns.

All these applications rely on feedback for their ability to learn and adapt. To see how such environmental feedback works, consider a simple thermostat. A bimetallic thermostat has a strip made of two metals fastened together that expand and contract at different rates when the temperature rises and falls. As a result, the thermostat bends when cold and straightens out when warmed (Fig. 13.1). When the temperature drops low enough, the thermostat bends far enough to close an electrical circuit that causes the heating to come on. When

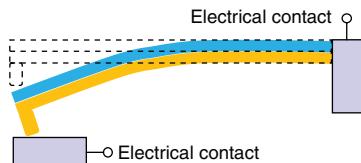


Fig. 13.1. Bimetallic thermostat made from iron (blue) and copper (orange). In cold, the copper contracts more and it bends the bimetal strip downward.

the temperature is too hot, the strip straightens, the circuit is broken, and the heating goes off. Nowadays, sensitive temperature sensors have replaced most metal thermostats, but the principle of how they control the heating system using feedback from the environment remains the same.

Wiener argued that although the physical sciences had been the dominant sciences of the past, the future would be more concerned with communication and control, and he believed that the computer would play a major role in such a future. He called his new science *cybernetics* from the Greek word *kybernetes*, meaning *steersman*. Wiener used the name to refer to the control of complex systems, but the prefix *cyber-* has acquired a variety of computer-related meanings. For example, we now talk about *cyberspace*, meaning the online world of computer networks, and *cyberwarfare*, for attacks on an enemy's information systems.

Scientists at a neurophysiology meeting in New York in 1942 took the first steps toward defining the field of artificial intelligence (AI). Wiener, with his colleagues Julian Bigelow and Arturo Rosenblueth, argued that an animal's nervous system could be thought of in engineering terms as a complicated network of *neurons*, the cells in the brain that process information, with feedback loops. They suggested we can think of computing systems in biological terms in the same way. It was through feedback, they concluded, that an engineering system could have a "definite purpose." This discussion marked the beginning of the fields of AI and *cognitive science* – the study of thinking, learning, and intelligence – although these terms were not introduced until more than a decade later. Cognitive science is now seen as bringing together computer modeling, neurophysiology, and psychology to try to understand how the human mind works.

Wiener and von Neumann were not the only ones thinking about the possibility of AI. In 1941, during World War II, Alan Turing had been exploring ideas about what he called "machine intelligence." Turing had helped design the *bombe*, a mechanical device used in the British code-breaking center at Bletchley Park to decrypt secret messages generated by the German Enigma machine. The bombe had demonstrated the value of performing "guided searches" to save time by reducing a large range of possible solutions to a manageable number. Turing and his fellow code breaker Donald Michie (B.13.2) had many discussions about how similar ideas could be used to create a computer chess program. In 1950, in his famous paper "Computing Machinery and Intelligence," he introduces the idea of what is now known as the *Turing Test*. In the Turing Test, if a human being cannot consistently tell whether questions are being answered by a computer or by another human being, then the computer has passed the test. In his paper, Turing considered the question "Can machines think?" He proposed replacing this question by another, more practical question based on what he called the *imitation game*. The essence of the game is that there are three people in different rooms – a man A, a woman B, and an interrogator C. The three people can communicate only by sending typewritten messages. The object of the game is for the interrogator C to determine whether A or B is the woman by asking questions of each of them. Turing now asks the question:

"What will happen when a machine takes the part of A in this game?" Will the interrogator decide wrongly as often when the game is played like this



B.13.2. Donald Michie (1923–2007) worked in the British code-breaking center at Bletchley Park during World War II. He was one of the pioneers of AI in the U.K. computer science research community.

as he does when the game is played between a man and a woman? These questions replace our original, “Can machines think?”³

The Turing Test is often taken as an operational definition of intelligence. In its modern form, it reads, “A computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or from a computer.”⁴ To pass the test, computers will need to complete the following tasks: to understand natural language; reason about the information expressed by words and sentences; and learn from experience. In 1950, Turing was cautiously optimistic:

I believe that in about fifty years' time it will be possible, to programme computers, with a storage capacity of about 10^9 , to make them play the imitation game so well that an average interrogator will not have more than 70 per cent chance of making the right identification after five minutes of questioning. The original question, “Can machines think?” I believe to be too meaningless to deserve discussion. Nevertheless I believe that at the end of the century the use of words and general educated opinion will have altered so much that one will be able to speak of machines thinking without expecting to be contradicted.⁵

Turing also gave a famous example of the type of conversation that he imagined it would be possible to have with a “sonnet-writing” machine in the future. It would be difficult to learn whether the machine has really understood something or whether, as he says, it has just “learnt it parrot fashion”:⁶

Interrogator: In the first line of your sonnet which reads “Shall I compare thee to a summer's day”, would not “a spring day” do as well or better?

Witness: It wouldn't scan.

Interrogator: How about “a winter's day”? That would scan all right.

Witness: Yes, but nobody wants to be compared to a winter's day.

Interrogator: Would you say that Mr. Pickwick reminded you of Christmas?

Witness: In a way.

Interrogator: Yet Christmas is a winter's day, and I do not think Mr. Pickwick would mind the comparison.

Witness: I don't think you are serious. By a winter's day one means a typical winter's day, rather than a special one like Christmas.⁷

If a computer were capable of such a sophisticated dialog, requiring knowledge both of literature and Mr. Pickwick as well as of the significance of Christmas, it would be hard to make a distinction between “real” and artificial thinking. At present, we still seem to be far from this goal. ELIZA, one of the earliest “chatbot” programs, simulated an interview with psychotherapist and could be superficially very convincing (see the short summary of ELIZA at the end of this chapter, for an example). Its author, Joseph Weizenbaum, chose the psychotherapy model precisely because it would not require a significant knowledge base. ELIZA imitated client-centered therapy, a form of psychotherapy that tries to increase the patient's insight and self-understanding by restating the patient's feelings and thoughts.



Fig. 13.2. The Loebner Prize for \$100,000 was established in 1990 for the AI system that first passes the Turing Test.



Fig. 13.3. CAPTCHAs can be easily read by a human, but not by a computer. This is one commonly used mechanism to distinguish between human visitors to websites and robotic crawlers.



B.13.3. Luis von Ahn is an associate professor at Carnegie Mellon University in Pittsburgh. He is perhaps best known for his invention of CAPTCHAs, those annoying distorted characters that only humans, not computers, are supposed to be able to read.

Since 1991, a New York businessman, Hugh Loebner (Fig. 13.2), has sponsored an annual Turing Test competition; and in 2012, the centenary of Turing's birth, the contest was held at Bletchley Park. In more than twenty years of competition, no chatbot program has come close to deceiving a sophisticated judge.

An everyday demonstration of a computer's inability to pass something like a Turing Test is a reverse version based on recognizing distorted letter shapes. To pass this reverse Turing Test, a computer would need highly developed perceptual abilities that are currently beyond the capability of the most advanced computer vision algorithms. These puzzles were called CAPTCHAs (Fig. 13.3) by Luis von Ahn (B.13.3), an acronym standing for "Completely Automated Public Turing test to tell Computers and Humans Apart." Humans can easily recognize the distorted letters, so CAPTCHAs enable websites to distinguish between human users and automated "robot" programs trying to access the site. It is estimated that more than two hundred million CAPTCHAs are solved every day.

From logic theorist to DENDRAL

The term *artificial intelligence* was coined by John McCarthy (B.13.4) in a workshop at Dartmouth College in New Hampshire in 1956. McCarthy and fellow AI pioneers Marvin Minsky, Claude Shannon, and Nathaniel Rochester wrote a proposal for the workshop stating:

The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves. We think that significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer.⁸

The highlight of the Dartmouth workshop was a reasoning program developed by Allen Newell and Herbert Simon (B.13.5) from Carnegie Tech, now Carnegie Mellon University. Their "Logic Theorist" program was able to prove theorems using simple symbolic logic. It represented each problem as a tree structure with the root being the initial *hypothesis*, a tentative explanation that could be tested by further investigation. Each branch of the tree was a deduction based on the rules of logic. To prevent the tree from growing uncontrollably, Newell and Simon needed a way to remove unwanted branches. To do so, they introduced *heuristics*, rules of thumb that enabled the program to select only those branches of the overall search tree that seemed most promising. They said, "Logic Theorist's success does not depend on the 'brute force' use of the computer's speed, but on the use of heuristic processes like those employed by humans."⁹

In their monumental work *Principia Mathematica*, Alfred Whitehead and Bertrand Russell had attempted to systematize the principles of mathematical



B.13.4. A famous photograph of four of the founding fathers of AI. From left to right they are Claude Shannon, John McCarthy, Ed Fredkin, and Joseph Weizenbaum.

logic. Newell and Simon attempted to use Logic Theorist to reproduce the proofs of fifty-two theorems in Whitehead and Russell's book:

Let us consider more specifically whether we should regard the Logic Theorist as creative. When the Logic Theorist is presented with a purported theorem in elementary symbolic logic, it attempts to find a proof. In the problems we have actually posed it, which were theorems drawn from [Chapter 2](#) of Whitehead and Russell's *Principia Mathematica*, it has found the proof about three times out of four.¹⁰

On being told that the program had found a shorter proof for one of their theorems, Bertrand Russell was reportedly delighted. Newell, Shaw, and Simon attempted – unsuccessfully – to publish their result in the *Journal of Symbolic Logic* with the Logic Theorist program listed as a co-author.

McCarthy moved from Dartmouth to MIT in 1958 and in the same year made three major contributions to computer science. One was the suggestion for time-sharing systems, as we have seen in [Chapter 3](#). A second was his invention of the Lisp programming language, an acronym derived from LISt Processing. Lisp was the dominant language for AI applications for the next thirty years. McCarthy's third major contribution was to lay out a research agenda for the AI community in a paper called "Programs with Common Sense." In the paper, McCarthy described a hypothetical AI program he called Advice Taker. Like Newell and Simon's Logic Theorist and their ambitious follow-up, the General Problem Solver, Advice Taker would not only use logic and *symbol manipulation*, the manipulation of characters rather than numbers, but also incorporate general knowledge about the world to assist in its deductive process:

The main advantages we expect the advice taker to have is that its behavior will be improvable merely by making statements to it, telling it about its symbolic



B.13.5. Herbert Simon (1916–2001) and Allen Newell (1927–92) were pioneers in the field of AI. They were awarded the Turing Award in 1975 for their work in AI, and Simon also won the Nobel Prize in economics in 1978 for his theory of decision making.

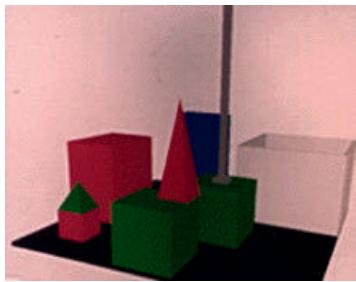


Fig. 13.4. The SHRDLU “blocks world” program was written by Terry Winograd at MIT. The program could understand and execute instructions given in natural language to move different types of blocks around in a virtual box.

environment and what is wanted from it. To make these statements will require little if any knowledge of the program or the previous knowledge of the advice taker. One will be able to assume that the advice taker will have available to it a fairly wide class of immediate logical consequences of anything it is told and its previous knowledge. This property is expected to have much in common with what makes us describe certain humans as having common sense. We shall therefore say that a program has common sense if it automatically deduces for itself a sufficiently wide class of immediate consequences of anything it is told and what it already knows.¹¹

Advice Taker embodied the idea that an AI system needed an explicit representation of the world and the ability to manipulate this knowledge with logical deductive processes. This vision set the agenda for AI research for the next few decades.

Marvin Minsky (B.13.6) arrived at MIT at the same time as McCarthy, and together they set up the MIT Artificial Intelligence Laboratory. Their research collaboration lasted only a few years before their approaches to AI diverged. Minsky concentrated on just getting systems to do interesting things – “scruffy AI.” Minsky’s students focused on solving problems in very limited domains, application areas not requiring a broad general knowledge. Successful examples included the domains of integral calculus, geometry, and algebra as well as a famous series of problems in the “blocks world” (Fig. 13.4), a simplified world consisting of some toy blocks sitting on a table. SHRDLU, developed by Terry Winograd, was a computer program that could understand instructions and carry on conversations about the blocks world. Unlike Minsky, McCarthy emphasized knowledge representations and reasoning using formal logic. In 1963 McCarthy left MIT to start the Stanford Artificial Intelligence Laboratory.

As computers became more powerful and as computer memories became larger, there was a movement for researchers to explicitly build “knowledge” into AI applications and to develop *expert systems*, computer programs that imitate the decision making of a human expert. One of the pioneers of the expert-systems approach to AI was Ed Feigenbaum (B.13.7) at Stanford University. In 1969, with Bruce Buchanan and Joshua Lederberg, a geneticist and recipient of the Nobel Prize, Feigenbaum developed the DENDRAL program that attempted to capture the expert knowledge of chemists and to apply that knowledge by employing a set of rules. The name DENDRAL was an abbreviation of *dendritic algorithm*, *dendritic* referring to the branching fibers of neurons that pick up nerve impulses. The problem that DENDRAL attempted to solve was that of determining the molecular structure of a substance using data provided by a mass spectrometer, an instrument that separates particles of different masses in a similar way to light spread out into different colors by a prism. To identify the precise structure of a compound, a chemist must deduce its chemical elements from the set of masses of fragments of the compound. For large molecules, this generates a huge number of possible structures. To make the problem manageable, expert chemists use their own rules of thumb – in other words, *heuristics* – to recognize well-known substructures and thereby reduce the number of possibilities for the overall structure of the compound. DENDRAL combined a knowledge base, written in the form of rules, with a reasoning engine written in Lisp. DENDRAL was



B.13.6. Marvin Minsky is one of the original pioneers of AI. He is also credited with the invention of the first head-mounted graphical display in 1963. He also acted as a scientific adviser for Stanley Kubrick’s film *2001: A Space Odyssey*. Science fiction writer Isaac Asimov described Minsky as one of only two people he would admit were smarter than he was. The other was cosmologist and astronomer Carl Sagan.

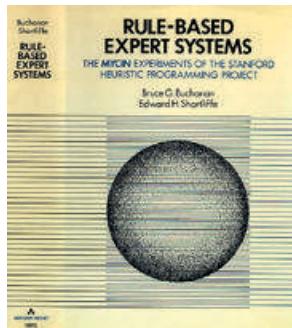


Fig. 13.5. The MYCIN project was an expert system designed to diagnose and treat blood infections. It was developed at Stanford by Edward Shortliffe with Bruce Buchanan and Ed Feigenbaum.

thus the first successful “knowledge-intensive” AI system because it automated the decision-making and problem-solving processes of experts in a field.

Feigenbaum looked at other domains where this approach could be applied. With Bruce Buchanan and Edward Shortliffe, he developed the MYCIN expert system to diagnose and treat blood infections (Fig. 13.5). Using about 450 rules developed from interviews with experts, MYCIN performed better than many doctors. The success of DENDRAL, MYCIN, and other expert systems led to an overenthusiastic rush to produce commercial systems in the 1970s and 1980s. Although the high hopes of the pioneers were not fully realized, knowledge-based expert systems are still used for applications ranging from straightforward help-desk and technical support to manufacturing and robotics. For narrow, well-defined problems, expert systems can be successful. However, a major limitation of this rule-based approach to knowledge is that these systems do not generalize well to larger, broader problems. In addition, the development and capture of the knowledge rules are very labor intensive and usually very specific to the case at hand. Because almost nothing in real life is simply true or false in the way that abstract logic requires, for any commonsense rule about the world there must also be a large number of exceptions.

The creation of taxonomies and classifications dates back to the 300s B.C., when Aristotle wrote his *Organon*, a collection of his works on logic. This included a section on categories that we would now call a type of *ontology*, the study of what kinds of things exist. It was the Swedish biologist Carolus Linnaeus who invented our present system of biological classification in the 1700s (Fig. 13.6). Computer scientists have borrowed the word *ontology* from the philosophers to describe a structural framework for organizing knowledge. An ontology specifies a set of concepts within a domain that a computer can use to reason about objects in the domain and about the relationships between them. AI researchers have long believed that useful ontologies are essential for effective AI systems. One response to this need is therefore to expand the knowledge base of the computer by producing a comprehensive vocabulary of all of the important concepts in a given domain, including the objects in the domain and the properties, relations, and functions needed to define the objects and specify their actions.

One of the most ambitious ontology projects is the Cyc project, started by Douglas Lenat in 1984. The name Cyc is a shortened form of *encyclopedia*. The project is an attempt to build a knowledge base containing much of the everyday, commonsense knowledge of a human being. Typical pieces of knowledge represented in the database are statements such as “Every tree is a plant” and “Plants die eventually.” After more than twenty-five years, Cyc’s knowledge base contains more than one million assertions, rules, or commonsense ideas. However, its creators estimate that it will need more than one hundred times that many entries before Cyc can begin to learn for itself from written material.

The DBpedia (Fig. 13.7) project has taken a different approach and uses a method called *crowdsourcing*, soliciting content from a large group of people, to extract structured data from Wikipedia. DBpedia’s 2012 release contained an ontology with more than two million concepts together with about one hundred facts per concept. The researchers hope that the Cyc and DBpedia projects will help realize Tim Berners-Lee’s vision of the Semantic Web, in which machines



B.13.7. Ed Feigenbaum received the Turing Award for his work in expert systems in 1994. He is often known as the “father of expert systems.” He also served as chief scientific adviser for the U.S. Air Force and received their Exceptional Civilian Service Award in 1997.

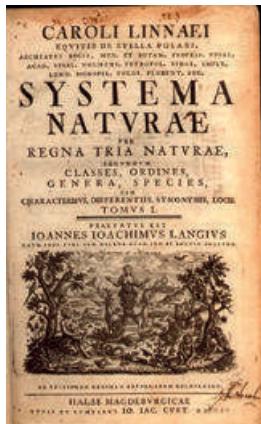


Fig. 13.6. Swedish botanist, physician, and zoologist, Carolus Linnaeus, published his classification of living things in 1735. This was an early attempt at constructing a knowledge representation for species of animals and plants.

can process and understand the actual data on the web. This vision will become reality when web search engines have access to machine-readable knowledge that enables them to reason and make “intelligent” decisions.

The early optimism of the Dartmouth workshop attendees – Allen Newell, Herbert Simon, John MacCarthy, and Marvin Minsky – was typified by the quotation that introduces this chapter. A more realistic perspective has now replaced this optimism. As the computer scientist David McAllester said in a 1998 paper on machine learning:

In the early period of AI it seemed plausible that new forms of symbolic computation ... made much of classical theory obsolete. This led to a form of isolationism in which AI became largely separated from the rest of computer science. This isolationism is currently being abandoned. There is a recognition that machine learning should not be isolated from information theory, that uncertain reasoning should not be isolated from stochastic modeling, that search should not be isolated from classical optimization and control, and that automated reasoning should not be isolated from formal methods and static analysis.¹²

Computer chess and Deep Blue

In the early days of computing, most people thought that computers were just machines that were capable of carrying out complex arithmetic calculations very rapidly. A few of the early pioneers, like Turing and Shannon, speculated that computers one day would be able to play chess, a task that had always up until then been considered to require human intelligence. Donald Michie summarized the interest of chess for AI as follows:

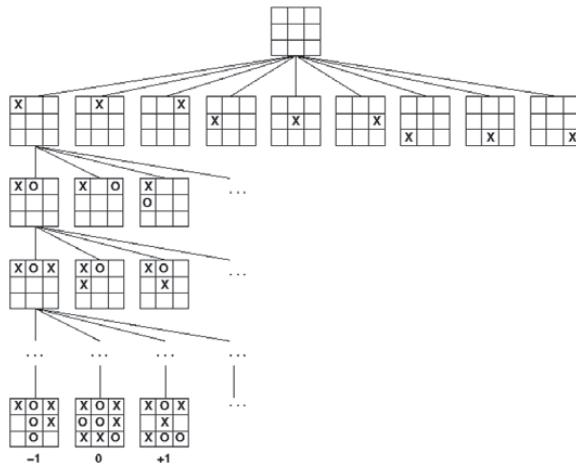
Computer chess has been described as the “Drosophila melanogaster” of machine intelligence. Just as Thomas Hunt Morgan and his colleagues were able to exploit the special limitations and conveniences of the “Drosophila” fruit fly to develop a methodology of genetic mapping, so the game of chess holds special interest for the study of the representation of human knowledge in machines. Its chief advantages are: (1) chess constitutes a fully defined and well-formalized domain; (2) the game challenges the highest levels of human intellectual capacity; (3) the challenge extends over the full range of cognitive functions such as logical calculation, rote learning, concept-formation, analogical thinking, imagination, deductive and inductive reasoning; (4) a massive and detailed corpus of chess knowledge has accumulated over the centuries in the form of chess instructional works and commentaries; (5) a generally accepted numerical scale of performance is available in the form of the U.S. Chess Federation and International ELO rating system.¹³

Claude Shannon’s 1950 article “Programming a Computer for Playing Chess” that spelled out a complete set of ideas for computer chess, including how to represent board positions, searching the “game tree” of possible moves, and using procedures called *evaluation functions*, by which players use knowledge of the game to judge each possible move and choose the best ones. In game theory, a *game tree* is a graphical representation of a sequential game consisting of



Fig. 13.7. The DBpedia project is trying to structure the content of Wikipedia by using an army of volunteers – ‘crowdsourcing’ – to perform the work required.

Fig. 13.8. A (partial) game tree for tic-tac-toe or noughts and crosses.



nodes, the points at which players can take actions, and *branches*, which represent the possible moves at each node. In 1951, Dietrich Prinz wrote the first chess program able to solve simple endgame problems. Prinz worked for Ferranti, a British computer company marketing the Manchester Mark I machine, the first commercially available general-purpose computer. Five years later, Stan Ulam and a group at Los Alamos National Laboratory wrote a program that could play a full game of chess, but only on a reduced board of 6×6 squares and no bishops. It was not until 1957 that IBM programmer Alex Bernstein wrote the first complete chess program for the IBM 704 computer, one of the last vacuum-tube computers. The chess program took about eight minutes to make a move after completing a search that could look about two moves ahead. Before we examine how a chess program works, let us look at a simpler problem, a computer program for the game called tic-tac-toe or noughts and crosses.

We shall label the two players MAX, who makes the X moves, and MIN, who makes the O moves. The total game tree consists of all the legal moves from all the possible configurations of Xs and Os. MAX moves first, and from the top node of the tree, MAX can make nine possible moves – a branching factor of nine (see Fig. 13.8). Then it is MIN's turn to make one of the eight remaining moves. This alternation continues until either a line of Xs or Os is achieved or all spaces on the board are filled. There are $9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1$ nodes, or 362,880 nodes, in the tree. The game has a simple evaluation function by which a player chooses the best move: +1 for a win for MAX, $\frac{1}{2}$ for a draw, and 0 for a win for MIN. A computer program can easily evaluate all possible paths and positions leading to the final move of the game.

For the first move in chess, there are twenty possible moves, sixteen with the eight pawns and four with the two knights. A typical game is around forty moves, and for each position there is an average of thirty to thirty-five possible moves to explore. Because the entire chess game tree would contain more than 10^{40} nodes, an exhaustive search strategy looking at all the final positions is not possible. Because we cannot get to the final positions, the evaluation function for chess is also much more complicated. For example, a chess evaluation function usually has a weighted sum of the various factors that are thought to

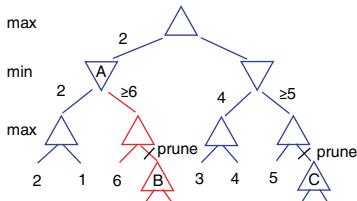


Fig. 13.9. In this example of alpha-beta pruning, the moves of the game are represented by a binary min-max tree. The algorithm traverses the tree starting from the bottom left and chooses the maximum of the first two values. The algorithm then moves to the red branch and finds that the first value is six. Thus what must be passed on to the min node must be more or equal to six. Because we already know that there is a lower value of two at the min node we do not need to evaluate branch B of the red node because the algorithm will never use this part of the tree. We proceed in the same way to eliminate branch C.

influence the value of a position. These include factors such as the power of each piece and its possible mobility, control of the center of the chess board, and the safety of the king. A program therefore needs to find strategies that optimize moves for the player MAX, at the same time assuming that the opponent, MIN, will make an optimum move in response. Such a strategy is provided by the *minimax* algorithm, a procedure that minimizes the risk for a player. Because the number of moves that need to be examined by the minimax algorithm grows at an increasingly rapid rate with the depth of the tree, computer chess programs can only afford to evaluate several moves ahead, not all the way to the final result node. In their 1958 chess program dubbed NSS from their initials, Allen Newell and Herbert Simon from Carnegie Mellon University and Cliff Shaw from the RAND Corporation introduced an optimization technique called *alpha-beta pruning* (Fig. 13.9) for the minimax search algorithm. Alpha-beta pruning decreases the search time by stopping evaluation of a move when at least one possibility has been found that proves the move to be worse than a previously examined move. In this way, several branches of the search tree can be “pruned” and the search time devoted to deeper exploration of more promising branches. In addition to such pruning techniques, modern chess programs also include tables of the standard openings and endgames.

The first computer versus computer chess match featured the Kotok-McCarthy program written by Alan Kotok, John McCarthy, and their colleagues from MIT pitted against the Russian ITEP program written by scientists at the Institute of Theoretical and Experimental Physics in Moscow (Fig. 13.10). Playing by telegraph in 1967, the match ended in a 3 to 1 victory for ITEP. In the same year, MIT's Mac Hack, written by Richard Greenblatt and colleagues, became the first chess program to play in a tournament with humans. Its Elo rating was 1400, well above the novice level of 1000 on the chess rating system developed by the Hungarian-born physicist Árpád Élő. In 1968, the international chess master David Levy made a famous bet with John McCarthy that no computer would beat him at chess in the next ten years, saying:

Clearly, I shall win my ... bet in 1978, and I would still win if the period were to be extended for another ten years. Prompted by the lack of conceptual progress over more than two decades, I am tempted to speculate that a computer program will not gain the title of International Master before the turn of the century and that the idea of an electronic world champion belongs only in the pages of a science fiction book.¹⁴



Fig. 13.10. A photograph of the Institute of Theoretical and Experimental Physics in Moscow.

Levy played his 1978 match against the Chess 4.7 program, the strongest computer chess program of the time, written by Larry Atkin and David Slate from Northwestern University. Levy won by 4.5 to 1.5 but he said later, “I had proved that my 1968 assessment had been correct, but on the other hand my opponent in this match was very, very much stronger than I had thought possible when I started the bet.”¹⁵

In 1980, the celebrated MIT computer scientist Ed Fredkin offered prizes for successive milestones in computer chess. The smallest prize of \$5,000 went to Ken Thompson, inventor of the Unix operating system, and Joe Condon, when their Belle chess program earned a U.S. Master rating in 1983. Belle was the first computer chess system to use custom-designed chips, and it won the



Fig. 13.11. IBM's Deep Blue chess computer first played Kasparov in 1996. On that occasion the world champion managed to beat the machine. Kasparov famously lost the rematch a year later.

world computer chess championship in 1980. The U.S. Department of State temporarily confiscated Belle in 1982 as it was heading to the Soviet Union to participate in a computer chess tournament. The State Department claimed it was a violation of U.S. technology transfer law to ship a high-technology computer to a foreign country. The next prize of \$10,000 for the first program to achieve an Elo rating of 2500 was awarded to a computer called Deep Thought in August 1989. Deep Thought was a computer specifically designed to play chess by Feng-hsiung Hsu and his fellow graduate student Murray Campbell at Carnegie Mellon University. IBM then recruited Hsu and Campbell to develop a successor to Deep Thought. The result was Deep Blue, a parallel computer with thirty processors, enhanced by 480 special-purpose chess chips (Fig. 13.11). Deep Blue was capable of evaluating two hundred million positions a second and could typically search six to eight moves ahead, and sometimes more. A team of three chess grand masters provided its opening library, and its end-game database included many six-piece endgames as well as those with five pieces and fewer. In May 1997, world champion Garry Kasparov took on Deep Blue in a six-game match held in New York (Fig. 13.12). The computer won a close match with two wins for Deep Blue, one for Kasparov, and three draws. The \$100,000 Fredkin Prize went to Feng-hsiung Hsu, Murray Campbell, and Joseph Hone from IBM. After the match, Kasparov wrote:

The decisive game of the match was Game 2, which left a scar in my memory ... we saw something that went well beyond our wildest expectations of how well a computer would be able to foresee the long-term positional consequences of its decisions. The machine refused to move to a position that had a decisive short-term advantage – showing a very human sense of danger.¹⁶

Neural networks

In the audience for Norbert Weiner's neurophysiology talk in 1942 was Warren McCulloch (B.13.8), a professor of psychiatry in Chicago. With a precocious eighteen-year-old mathematician called Walter Pitts, McCulloch developed the first model of the brain as an electrical network of interconnected neurons. They argued that their idealized "neural network" model captured the key features of the brain's physiology. Von Neumann was so impressed by this work that, together with Wiener and Howard Aiken from Harvard, he organized a small workshop at Princeton in January 1945 at which McCulloch and Pitts were invited to present their neural network model. Ideas from neural networks were fresh in von Neumann's mind when he wrote his "Draft Report on the EDVAC" later that year – in which he referred to the basic functional units of the computer as "organs" and made comparisons of the functions of these units with the biological functions of neurons.

The importance of the brain in determining human emotions was recognized by Hippocrates, the "Father of Medicine," as long ago as 400 B.C. He said, "Men ought to know that from nothing else but the brain come joys, delights, laughter and sports, and sorrows, griefs, despondency, and lamentations."¹⁷ The human brain has a similar structure to brains of other mammals but is significantly larger in relation to body size compared to most animals. The relative increase in size of the human brain is mainly due to the greater size of



Fig. 13.12. The newspapers and other media portrayed the 1997 match between World Chess Champion Garry Kasparov and IBM's Deep Blue computer as a battle between human and machine. The cover of Newsweek proclaimed it "The Brain's Last Stand."

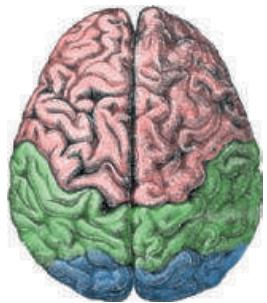


Fig. 13.13. A diagram of the cerebral lobes of a human brain: frontal lobes in pink, parietal lobe in green, and the occipital lobe in blue.

the *cerebral cortex*, a thick layer of neural tissue that covers most of the brain (see Fig. 13.13). The name *cortex* comes from the Latin word for the bark of a tree, but in this case it means the outer layer of an organ. The cortex is deeply folded and ribbed because such folding maximizes the amount of brain surface that can fit into the limited space of the skull. More than two-thirds of the surface area of a human brain is buried in these folds, called *sulci*. The cerebral cortex plays a key role in memory, perception, thought, language, and consciousness.

The nineteenth century brought rapid progress in biological science thanks to the wide use of microscopes. Theodor Schwann and Matthias Schleiden had suggested cell theory, according to which all living organisms are made up of cells, in 1838. But not all scientists were convinced that cell theory applied to brain tissue. As a result, many scientists experimented with different chemical substances for coloring the brain tissue so that individual cells would be made visible. A Spanish physician, Santiago Ramón y Cajal improved on a cell-staining method originally developed by the Italian doctor Camillo Golgi, and used this new technique to investigate the central nervous system of many living creatures. It was Ramón y Cajal's work that first revealed the complexity of biological neural networks. He wrote:

What beauty is shown in the preparations obtained by the precipitation of silver dichromate deposited exclusively onto the nervous elements! But, on the other hand, what dense forests are revealed, in which it is difficult to discover the terminal endings of its intricate branching.... Given that the adult jungle is impenetrable and indefinable, why not study the young forest, as we would say in its nursery stage.¹⁸

We now know that neurons consist of a cell body or *soma* with two types of nerve fiber growing from the cell, *dendrites* and *axons*. The cell body contains the genetic information and the molecular machinery required for the functioning of the neuron. The role of the dendrites is to receive electrical or chemical signals from other neurons and provide the input to the cell of the neuron. The axon, usually much longer than the dendrites, carries nerve impulses from the cell body to other neurons. Ramón y Cajal also suggested that these signals always flow in one direction, from the dendrites of the cell to the axon, and that the axon is connected to dendrites of other cells by structures called *synapses* (see Fig. 13.14). The word *synapse* comes from the Greek words *syn*, meaning *together*, and *haptein*, meaning *to clasp*. Golgi and Ramón y Cajal were awarded the 1906 Nobel Prize in physiology or medicine “in recognition of their work on the structure of the nervous system.”¹⁹

The number of neurons in the brain varies widely from species to species. The human brain is believed to contain more than eighty-five billion neurons, while the brain of a cat has only one billion and a chimpanzee about seven billion neurons. In addition to these vast numbers of neurons, the brain has an even larger number of synapses. Each human neuron has, on average, seven thousand synaptic connections to other neurons. There are many different types of neurons, and we will describe only how a “typical” neuron functions. The incoming signals reaching a neuron from all of its dendrites are collected and processed inside the cell body. Any output signal resulting from this input



B.13.8. Warren McCulloch (1898–1969) was an early pioneer of AI. With Walter Pitts he proposed the first mathematical model of a neural network. John von Neumann was very impressed by the McCulloch-Pitts model and the paper and its physics terminology influenced him as he wrote the EDVAC draft report.

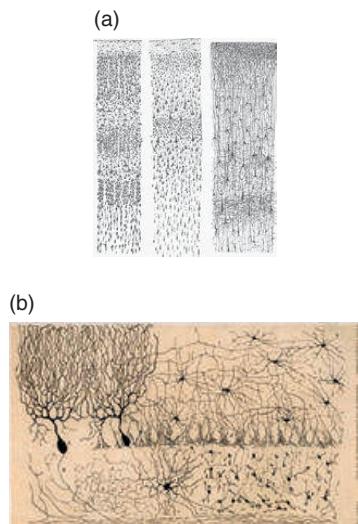


Fig. 13.14. Santiago Ramón y Cajal's drawing of: (a) a Golgi-stained cortex of a six-week-old human infant and (b) cells of the chick cerebellum.

travels down the axon and is passed on to the dendrites of neighboring neurons through the synapses (see Fig. 13.15). A typical neuron operates on a “threshold” or “all-or-none” principle meaning that the input stimulation, represented by the sum of all the incoming signals, must be above a certain threshold for the cell to produce an output signal.

The cerebral cortex consists of up to six horizontal layers of neurons and is about 2.5 millimeters or one-tenth of an inch thick. The neurons in each of these layers connect vertically to neurons in adjacent layers. With these new discoveries about the brain in the first half of the twentieth century, Nobel Prize recipient Charles Sherrington poetically imagined how the workings of the brain would look as it woke up from sleep:

The great topmost sheet of the mass, that where hardly a light had twinkled or moved, becomes now a sparkling field of rhythmic flashing points with trains of traveling sparks hurrying hither and thither. The brain is waking and with it the mind is returning. It is as if the Milky Way entered upon some cosmic dance. Swiftly the head mass becomes an enchanted loom where millions of flashing shuttles weave a dissolving pattern, always a meaningful pattern though never an abiding one; a shifting harmony of subpatterns.²⁰

As we have seen, Wiener, von Neumann, Turing, and other early computer pioneers were fascinated with the possibility of computers performing operations that would normally be classified as requiring intelligence. Warren McCulloch and Walter Pitts had produced a simple mathematical model of a neuron that only “fired” when the combination of its input signal exceeded a certain threshold value (see Fig. 13.16). In their famous 1943 paper “A Logical Calculus of the Ideas Immanent in Nervous Activity,” they showed that a network of such neurons could carry out logical functions. They also suggested that, much like a human brain, these artificial neural networks (ANNs) could learn by forming new connections and by modifying the neural thresholds. Alan Turing put forward similar ideas in an unpublished paper on “Intelligent Machinery” in 1948. Turing suggested, “The cortex of an infant is an unorganised machine, which can be organised by suitable interfering training.”²¹

The basis of modern ANNs is a mathematical model of the neuron called the *perceptron* introduced by Frank Rosenblatt in 1957. In the original model

Fig. 13.15. Sketch of a biological neural network showing dendrites, axons, and synapses.

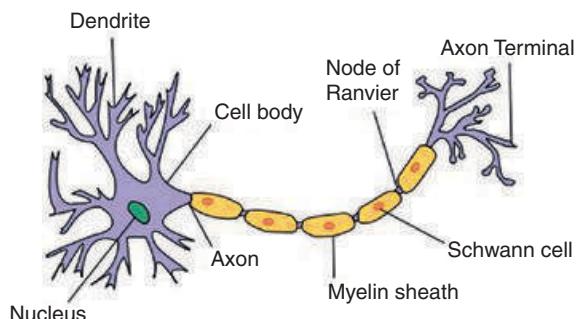


Fig. 13.16. Representation of an artificial neuron with inputs, connection weights, and the output subject to a threshold function.

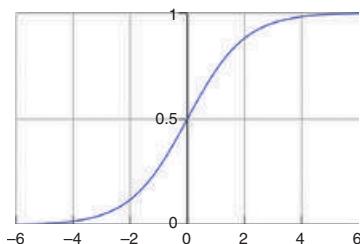
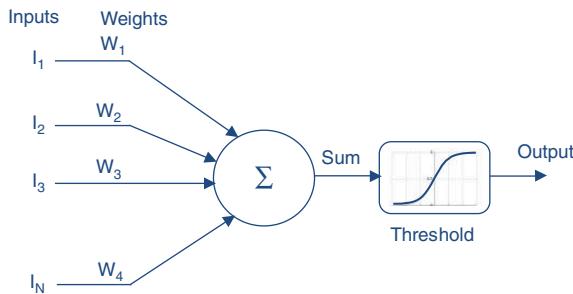


Fig. 13.17. A simple threshold function for an artificial neuron. The strength of the output signal depends on the magnitude of the sum of the input signals.

of McCulloch and Pitts, the input could only be either 0 or 1. In addition, each input “dendrite” had an associated “weight” that was either +1 or -1 to represent inputs that tended either to excite the neuron to fire or to inhibit the neuron from firing, respectively. The model calculated the weighted sum of the inputs – the sum of each input multiplied by its weight – and checked whether this sum was greater or smaller than the threshold value. If the weighted sum was greater than the threshold, the model neuron fired and emitted a 1 on its axon. Otherwise, the output remained 0. Rosenblatt’s perceptron model allowed both the inputs to the neurons and the weights to take on any value. In addition, the simple activation threshold was replaced by a smoother *activation function*, a mathematical function used to transform the activation level of the neuron into an output signal, such as the function shown in Figure 13.17. ANNs are just interconnected layers of perceptrons as shown in Figure 13.18.

For numerical calculations, computers are very much faster than humans at performing arithmetic. For tasks involving *pattern recognition* – the automatic identification of figures, shapes, forms, or patterns to recognize faces, speech, handwriting, objects, and so on – even young children are still very much better than the most powerful computers. The hope for ANN research is that by mimicking how our brains learn, these artificial networks can be trained to recognize patterns. The study of ANNs is sometimes called *connectionism*.

The publication of a famous book *Perceptrons* in 1969 by Marvin Minsky and Seymour Papert from MIT dashed early hopes for progress with neural networks. Minsky and Papert showed that a simple two-layer perceptron network was incapable of learning some very simple patterns. While they did not rule out the usefulness of multilayer perceptron networks with what they called “hidden” layers, they pointed out “the lack of effective learning algorithms”²² for such networks. This situation changed in the 1980s with the discovery of just such an effective learning algorithm. A very influential paper in the journal *Nature* gave the algorithm its name: “Learning Representations by Back-Propagating Errors” by David Rumelhart, Geoffrey Hinton (B.13.9), and Ronald Williams. Let us see how this *back-propagation* algorithm enables neural networks to learn.

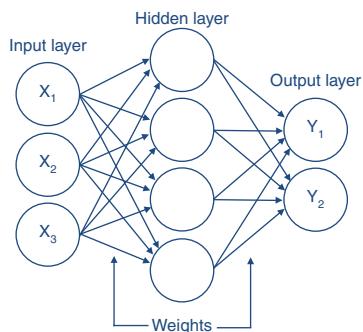


Fig. 13.18. An example of a three-layer ANN, with all connections between layers. The output of the neural network is specified by the connectivity of the neurons, the weights on the connections, the input signals, and the threshold function.



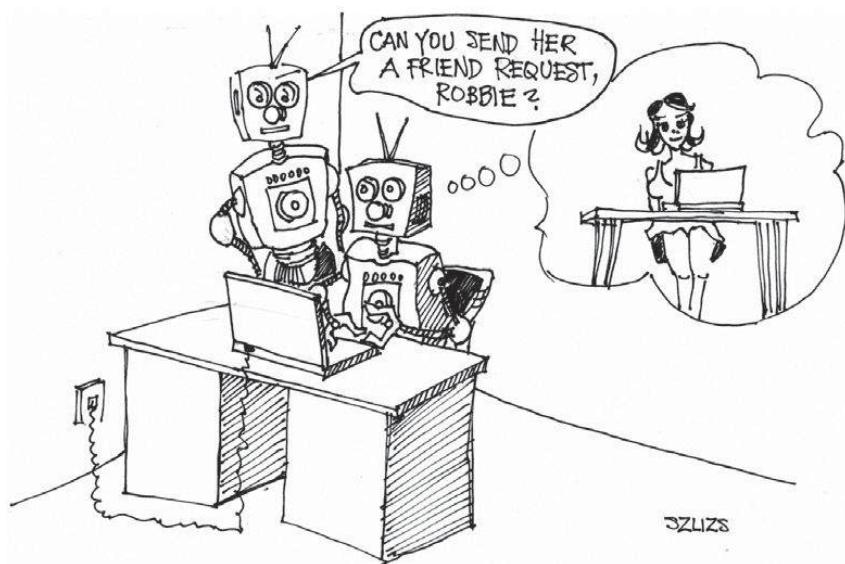
B.13.9. Geoffrey Hinton is a computer scientist based in Toronto who was one of the first computer scientists to show how to make computers “learn” more like a human brain. He has recently participated in exciting advances using so-called deep neural networks. His start-up company on such approaches to computer learning and recognition problems was bought by Google in 2013. Hinton is the great-great-grandson of logician George Boole. Photo by Emma Hinton

Imagine a simple three-layer neural network: a layer of input neurons, connected to a second hidden layer of neurons, which in turn is linked to a layer of output neurons (Fig. 13.18). Each neuron converts its inputs into a single output, which it transmits to neurons in the next layer. The conversion process has two stages. First, each incoming signal is multiplied by the weight of the connection, and then all these weighted inputs are added together to give a total weighted input. In the second stage, this combined input is passed through an activation function, such as the function of Figure 13.17, to generate the output signal for the neuron. To train the network to perform a particular task, we must set the weights on the connections appropriately. The amount of weight on a connection determines the strength of the influence between the two neurons. The network is trained by using patterns of activity for the input neurons together with the desired pattern of activities for the output neurons. After assigning the initial weights randomly, say to be between -1.0 to +1.0, then, by calculating the weighted input signals and outputs of the neurons in each layer of the network, we can determine the strength of the signals at the output neurons. For each input pattern, we know what pattern we want to see at the output layer, so we can see how closely our model output matches the desired output. We now have to adjust each of the weights so that the network produces a closer approximation to our desired output. We do this by first calculating the error, defined as the square of the difference between the actual and desired outputs. We want to change the weight of each connection to reduce this error by an amount that is proportional to the rate at which the error changes as the weight is altered. We first make such changes for all the neurons in the output layer. We then repeat the calculation to find the sensitivity to the weights connecting each layer, working backward layer by layer from the output to the input. The idea is that each hidden node contributes some fraction of the error at each of the output nodes to which it is connected. This type of network is known as a *feed-forward* network because the signals between the neurons travel in only one direction, from the input nodes, through the hidden nodes, to the output nodes. The learning algorithm to train the neural network is called *back propagation* because the error at the output layer is *propagated* – that is, passed along – backward through the hidden layer of the network (see Fig. 13.18).

In the 1990s, researchers found it hard to train neural networks with more than one hidden layer and two layers of weights. The problem was that the weights on any extra layers could not be adapted to produce significant improvements in learning. However, within the last few years, Geoffrey Hinton and colleagues from the University of Toronto, and researchers Li Deng and Dong Yu from Microsoft Research have shown that much deeper layered networks can not only be trained efficiently but can also deliver significantly improved learning outcomes. This *deep learning* approach is currently causing great excitement in the machine-learning community and is already leading to new commercial applications.

Key concepts

- Feedback loop
- Turing Test
- Expert systems
- Game tree
- Artificial neural network



ELIZA

The quest for a program that can imitate a human dates from the earliest days of computing. One of the earliest “chatbot” programs was written by Joseph Weizenbaum at MIT in 1964. The program was called ELIZA and was designed to create the impression that the user was having a conversation with a psychotherapist. Weizenbaum named the program ELIZA after Eliza Doolittle, a working-class character who learns to speak with an upper-class accent in George Bernard Shaw’s play *Pygmalion*. ELIZA uses an open-ended style of questioning similar to the client-centered psychotherapy introduced by psychologist Carl Rogers in the 1940s. This style of therapy aims to increase a patient’s understanding and insight by asking questions about the person’s thoughts and feelings and by rephrasing the responses. ELIZA has no understanding of the meaning of the conversation, but the program does “pattern matching” and follows some simple rules for formulating questions. The style of conversation that results can be seen from the following example script from an ELIZA session:

Person: Men are all alike.
ELIZA: In what way?
Person: They’re always bugging us about something or other.
ELIZA: Can you think of a specific example?
Person: Well, my boyfriend made me come here.
ELIZA: Your boyfriend made you come here?
Person: He says I’m depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
Person: It’s true. I am unhappy.
ELIZA: Do you think coming here will help you not to be unhappy?
Person: I need some help, that much seems certain.
ELIZA: What would it mean to you if you got some help?
Person: Perhaps I could learn to get along with my mother.
ELIZA: Tell me more about your family.

Weizenbaum has said that ELIZA provided a parody of the responses of a nondirectional psychotherapist in an initial psychiatric interview. He also said that he chose the context of psychotherapy to “sidestep the problem of giving the program a data-base of real-world knowledge”²³ because the therapeutic situation is one of the few human situations in which a human being can reply to a statement with a question that indicates very little specific knowledge of the topic under discussion. The dialog could sometimes be so convincing that some users thought they were dealing with a human therapist instead of a machine, and there are many anecdotes about people becoming emotionally engaged with the ELIZA program.

I4 Machine learning and natural language processing

... one naturally wonders if the problem of translation could conceivably be treated as a problem in cryptography. When I look at an article in Russian, I say "This is really written in English, but it has been coded in some strange symbols."

Warren Weaver¹

Ideas of probability: The frequentists and the Bayesians

We are all familiar with the idea that a fair coin has an equal chance of coming down as heads or tails when tossed. Mathematicians say that the coin has a probability of 0.5 to be heads and 0.5 to be tails. Because heads or tails are the only possible outcomes, the probability for either heads or tails must add up to one. A coin toss is an example of *physical probability*, probability that occurs in a physical process, such as rolling a pair of dice or the decay of a radioactive atom. Physical probability means that in such systems, any given event, such as the dice landing on snake eyes, tends to occur at a persistent rate or relative frequency in a long run of trials. We are also familiar with the idea of probabilities as a result of repeated experiments or measurements. When we make repeated measurements of some quantity, we do not get the same answer each time because there may be small random errors for each measurement. Given a set of measurements, classical or *frequentist* statisticians have developed a powerful collection of statistical tools to estimate the most probable value of the variable and to give an indication of its likely error.

An alternative view of probability reflects the strength of our belief that the coin is fair and not what statisticians call *biased*, tending to give one result more frequently than the other. For example, perhaps we have reason to think that the coin being fair, with a 50 percent probability of coming up heads, is only one possibility. Maybe we think there is an equal chance that the coin is biased and will come up as heads 80 percent of the time. Before tossing the coin, we suppose that each of these two options is equally likely. But after tossing the coin ten times and observing eight heads, we will want to modify our beliefs. It now makes sense for us to believe that there is a greater than fifty-fifty chance

that the coin is biased toward heads. The assumptions we had before tossing the coin are called *prior beliefs* – or just *priors* – because we form them prior to gathering any evidence about the situation. After incorporating the results of observations, we modify our beliefs. These modified beliefs are called *posterior beliefs* or *posteriors*. The technology of *Bayesian inference* that we explain in this section determines – in a precise, mathematical way – how we should change our prior beliefs. Bayesian inference is a decision-making technique that takes into account both observed data and prior beliefs and allows us to eliminate the least likely options. The frequentist interpretation of probability is fine for problems where we can make repeatable experiments and measurements. But such a frequentist approach cannot make predictions about nonrepeatable events, such as “What is the probability of an earthquake in Seattle next year?” or, more commonly, “What is the probability of rain in Seattle tomorrow?” The Bayesian approach can provide a mathematical basis for such predictions.

An English clergyman named Thomas Bayes (B.14.1) introduced what we now call the Bayesian approach in the 1700s. His goal was to learn the probability of a future event given only the number of times such an event had or had not occurred in the past. His paper “An Essay towards Solving a Problem in the Doctrine of Chances” contains the following example:

Picture a newborn witnessing his first sunset. Being new to this world, he doesn’t know if the sun will rise again. Making a guess, he gives the chance of a sunrise even odds and places in a bag a black marble, representing no sunrise, and a white marble, representing a sunrise. As each day passes, the child places in the bag a marble based on the evidence he witnesses – in this case, a white marble for each sunrise. Over time, the black marble becomes lost in a sea of white, and the child can say with near certainty that the sun will rise each day.²

This example illustrates the basic Bayesian approach. The newborn has an initial degree of belief in whether or not the sun will rise that is just a fifty-fifty guess. This belief is the baby’s *prior*. As the child gathers more data, he or she can update this belief to obtain a more accurate prediction for the probability of a sunrise, the *posterior*.

In his paper, Bayes describes a *thought experiment* in laborious detail, an experiment that we could now simulate very easily with a computer. He imagines that he turns his back to a square table and asks his assistant to throw a ball onto the table. The ball has just as much chance of landing at any place on the table as anywhere else. Bayes cannot see the table and has no idea where



B.14.1. Thomas Bayes (1701–61) was an English clergyman who did pioneering work in probability theory. In fact, his major work was published after his death and his papers were edited by the Welsh scientist, clergyman, and philosopher William Price. Bayes’ paper, “An Essay towards Solving a Problem in the Doctrine of Chances,” containing the famous result now known as Bayes Theorem, was published in *Philosophical Transactions of the Royal Society* in 1763. William Price had a remarkable career, was a personal friend of many of the founding fathers of the United States, and, with George Washington, received an honorary doctorate from Yale in 1781.

the ball has landed. The assistant now throws a second ball onto the table and reports to Bayes only that it landed to the left or right of the original ball. If it landed to the left, Bayes can deduce that the initial ball is slightly more likely to be in the right half of the table than in the left. The friend tosses another ball and reports that it lands to the right of the first ball. From this information, Bayes knows that the original ball cannot be at the extreme right of the table. With more and more throws, Bayes can narrow down the range of positions for the first ball and assign relative probabilities for different ranges. Bayes showed how it was possible to modify his initial guess for the position of the first ball, his prior probability, and to produce a new posterior probability by taking into account the additional data he had been given.

Although Bayes was first to suggest the use of probability to represent belief, it was the French mathematician Pierre-Simon Laplace (B.14.2) who developed this idea into a useful tool for many different types of problems. Laplace had become interested in probability through reading a book on gambling and did not initially know of Bayes' work. Laplace published his first paper on this subject in 1774 with the title "Mémoire sur la probabilité des causes par les événements" (Memoir on the Probability of the Causes of Events), so that his method is often abbreviated as just the "probability of causes." One of the first major applications of his new theory was to an analysis of the data on births in London and Paris. He wanted to know whether the data supported the suggestion of Englishman John Graunt that slightly more boys were born than girls. Using the christening records from London and Paris, Laplace concluded that he was "willing to bet that boys would outnumber girls for the next 179 years in Paris and for the next 8,605 years in London."³ Later in his life, Laplace turned to frequentist techniques to deal with the large quantities of reliable data on all sorts of subjects. In 1810, he proved what is now called the *central limit theorem*, which justifies the taking of the average of many measurements to arrive at the most probable value for a quantity. When the French government published detailed data on such events as thefts, murders, and suicides, all the governments in Europe started studying statistical data on a whole range of subjects. Bayes' idea that the probability of future events could be calculated by determining their earlier frequency was lost in a welter of numbers. As the nineteenth century progressed, few people regarded the idea that the uncertainty of some prediction could be modified by something as subjective as "belief" as a serious scientific approach. Apart from a few isolated instances, it was not until the middle of the twentieth century that mathematicians and scientists again took seriously a Bayesian interpretation of probability and considered it a valid tool for research. Today, the Bayesian approach has a wide variety of uses. For example, doctors employ it to diagnose diseases, and genetic researchers use it to identify the particular genes responsible for certain traits.



B.14.2. Pierre-Simon Laplace (1749–1827) was one of the giants of mathematics and science. He is often referred to as the "French Newton" because his works made a major contribution to many areas of knowledge including astronomy, mechanics, calculus, statistics, and philosophy. One of his tasks as a member of the French Academy was to standardize European weights and measures and, in 1799, the meter and the kilogram were introduced as standards.

Bayes' Rule and some applications

The modern revival of Bayesian thinking began in the 1940s. In 1946, Richard Cox, a physicist at Johns Hopkins University, looked again at the fundamentals of the Bayesian view of probability. In particular, he wanted to find

a consistent set of rules for reasoning about beliefs. First, he had to decide how to rank degrees of belief, such as whether the coin we talked about earlier was a fair coin with a 50 percent probability of coming up heads or a biased coin with an 80 percent probability of coming up heads. He proposed ranking how much we believe these possibilities by assigning a real number to each proposition such that the larger the number, the more we believe the proposition. He put forward two *axioms* (established rules) as being necessary for logical consistency. The first was that if we specify how much we believe something is true, we are also implicitly specifying how much we believe it is false. Using a scale of real numbers from 0 to 1 to specify beliefs, this axiom says that the belief that something is true plus the belief that the same thing is false must add up to one. This is the same as the usual *sum rule* for probabilities, which holds that the probabilities for all possible outcomes must add up to one.

Cox's second axiom is more complicated. If we specify how much we believe proposition Y is true, and then state how much we believe proposition X is true given that Y is true, then we must implicitly have specified how much we believe that both X and Y are true. Assuming some initial background information that we denote by B, we can write this belief relationship as an equation as follows:

$$\text{Prob}(X \text{ and } Y | B) = \text{Prob}(X | Y \text{ and } B) \times \text{Prob}(Y | B)$$

In words, this equation says that the probability that both X and Y are true, given background information B, is equal to the probability that X is true given that Y and B are true, times the probability that Y is true given B, regardless of proposition X. The vertical bar “|” separates the different propositions in these probabilities. This equation is the usual *product rule* for probabilities, which states that the probability of two independent events occurring simultaneously is the result of multiplying the individual probabilities together. The product rule is easy to derive from a frequentist approach. Note that all probabilities are conditional on the same background information B.

We can now derive the mathematical formula representing Bayes' Rule for probabilities. It is obvious that the probability that X and Y are both true does not depend on the ordering of X and Y on the left-hand side of our equation. We therefore have:

$$\text{Prob}(X \text{ and } Y | B) = \text{Prob}(Y \text{ and } X | B)$$

By expanding each side and doing a little rearrangement, we arrive at Bayes rule:

$$\text{Prob}(X | Y \text{ and } B) = \text{Prob}(Y | X \text{ and } B) \times \text{Prob}(X | B) / \text{Prob}(Y | B)$$

Put into words, Bayes' Rule states that the probability of your initial estimate X, given the original data B and some new evidence Y, is proportional to the probability of the new evidence, given the original data B and the assumption X, and to the probability of the estimate X, based only on the original data B. For example, the probability of drawing an ace from a deck of cards is 0.077

(4 cards divided by 52). If two cards are drawn at random, the probability of the second card being an ace depends on whether the first card was an ace. If it was, then the probability of the second card being an ace is 0.058 (3 divided by 52). If it wasn't, then the probability remains at 0.077.

Cox showed that quantifying beliefs numerically and requiring logical and consistent reasoning leads to exactly the same rules for beliefs as for physical probabilities. So there was no dispute about the validity of Bayes' Rule between frequentists and Bayesians. Instead, the controversy was about using subjective beliefs in data analysis rather than just using frequentist probabilities. The importance of Bayes' Rule for data analysis is apparent if proposition X is a *hypothesis* – that is, an idea or explanation – and Y is experimental *data*:

$$\text{Prob (hypothesis | data and B)} \sim \text{Prob (data | hypothesis and B)} \times \text{Prob (hypothesis | B)}$$

The symbol \sim means that the left-hand side is proportional to the right-hand side of the equation. In other words, the probability of the hypothesis being true given the data is proportional to the probability that we would have observed the measured data if the hypothesis was true. The second factor on the right-hand side is the probability of our hypothesis, **Prob (hypothesis | B)**. This is the prior probability and represents our state of belief before we have included the measured data. By Bayes' Rule, we see that this prior probability is modified by the experimental measurements using the quantity **Prob (data | hypothesis and B)** – known as the *likelihood function*. This gives us the posterior probability, **Prob (hypothesis | data and B)**, which is our new belief in the hypothesis, after taking into account the new data. The likelihood function uses a statistical model that gives the probability of the observed data for various values of some unknown parameter. For estimating the parameters of a model we can ignore the denominator, **Prob (data | B)**, because it is just a scaling factor that does not depend explicitly on the hypothesis. However, for model comparisons, the denominator is important and is called the *evidence*.

As Sharon McGayne shows in her book *The Theory That Would Not Die*, Bayesian reasoning about uncertainty persisted in some unlikely places, even in times when the frequentists were in the ascendancy. In 1918, Albert Whitney, who had taught insurance mathematics at the University of California, Berkeley, invented *credibility theory*, a Bayesian method for pricing insurance premiums by assigning weights to the available evidence based on its believability. In the 1930s, Cambridge geophysicist Harold Jeffreys studied earthquakes and tsunamis from a Bayesian point of view and published a classic text on the *Theory of Probability* in 1939, just before World War II broke out.

During World War II, a Bayesian approach to uncertainty played a decisive role in winning the battle against the German U-boats that were sinking vital U.K. supply ships in the North Atlantic. Alan Turing, working at the top-secret Bletchley Park code-breaking site, introduced Bayesian methods to help decipher the German Navy's messages encrypted using the Enigma machine. Soon after his arrival at Bletchley Park, Turing helped automate the process of searching through the huge number of possible Enigma settings. With

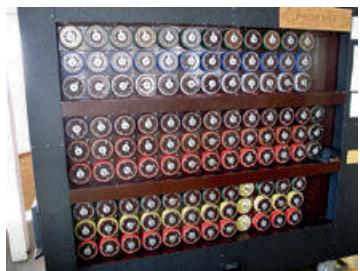


Fig. 14.1. A functioning replica of the *bombe* code-breaking machine rebuilt at Bletchley Park.

mathematician Gordon Welchman and engineer Harold Keen, Turing designed a machine called the *bombe*, a high-speed electromechanical machine that tested possible Enigma wheel arrangements (Fig. 14.1), saving time by reducing the number of possible solutions. However, in the worst case it could still take up to four days to try all the 336 possible wheel positions, and such a delay meant that the information was of no use for rerouting ships away from the U-boat packs. In an attempt to reduce the number of wheel positions that needed to be searched, Turing and the team looked for what they called *cribs*, German words that they thought likely to occur in the unencrypted text. Many of these came from German weather ships, which often repeated phrases like “weather for the night” and “beacons lit as ordered.” In addition, because the German operators spelled out numbers, the word *ein* (one) appeared in 90 percent of Enigma messages. Armed with such cribs, Turing invented a manual system that could reduce the number of wheel settings to be tested by the bombs from 336 to as few as 18. He called his system *Banburismus*, after the nearby town of Banbury where a printing shop produced the six-foot strips of cardboard needed to put his system into practice. To use the Banburismus system, staff at Bletchley punched holes corresponding to each intercepted message into a Banbury sheet. By putting one of the sheets on top of another, they could see when letter holes coincided on both sheets. This enabled Turing’s team to guess a stretch of letters. They could update this guess as more data arrived, using Bayesian inference, which uses a combination of new information and prior beliefs to eliminate the least likely choices. To compare the probabilities of his guesses, Turing introduced a unit of measurement he called a *ban* – short for Banburismus. A tenth of a ban was called a *deciban* and, according to Jack Good, Turing’s colleague at Bletchley, “A deciban is about the smallest weight of evidence perceptible to the intuition.”⁴ By June 1941, the Bletchley Park team could read messages to the German U-boats within an hour of their arrival.

After the war, the applications of Bayesian inference multiplied rapidly. Jerome Cornfield, working at the U.S. National Institutes of Health (NIH), introduced Bayesian methods into epidemiology, the branch of medicine that studies the incidence and distribution of diseases. Using Bayes’ Rule, Cornfield combined research showing the probability that someone with lung cancer was a cigarette smoker with NIH data to answer the opposite question, “What is the probability that someone who smokes will develop lung cancer?” His results showed that smokers are many times more likely to develop lung cancer than nonsmokers. At the RAND Corporation in Santa Monica, California, Fred Iklé and Albert Madansky used a Bayesian approach to estimate the probability of an accident involving nuclear weapons. Because there had only been “harmless” accidents with nuclear bombs, there was nothing that could be said about this question from a frequentist viewpoint. Their report was completed in 1958 but remained classified for more than forty years. However, the report persuaded General Curtis LeMay of the Strategic Air Command to issue orders that two people should be required to arm a nuclear weapon and also that combination locks should be installed on the warheads. A third application area was in business, where executives routinely have to make critical decisions with incomplete data and much uncertainty. At the Harvard Business School, Robert Schlaifer and Howard Raiffa introduced Bayesian methods to

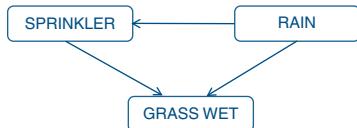


Fig. 14.2. A simple Bayesian network showing the structure of the joint probability distribution for rain, sprinkler, and grass. The diagram captures the fact that rain influences whether the sprinkler is activated, and both rain and the sprinkler influence whether the grass is wet. This is an example of a directed acyclic graph.

decision theory. Their 1961 book, *Applied Statistical Decision Theory*, contained a detailed account of Bayesian analytical methods.

The modern recognition of how Bayesian and frequentist methods can work together began in the 1980s. In 1984, Adrian Smith, a professor of statistics at the University of Nottingham in England, wrote that “efficient numerical integration procedures are the key to more widespread use of Bayesian methods.”⁵ Six years later, with Alan Gelfand from the University of Connecticut, Smith wrote a very influential paper showing that the difficult calculations required to apply Bayesian methods to realistic problems could be estimated using the *Monte Carlo method*. The Monte Carlo method is a forecasting technique applied in situations where statistical analysis is too difficult due to the complexity of the problem. As we have seen in [Chapter 5](#), the method involves running multiple trials using random variables: the larger the number of trials, the better the predictions work. A technique related to the Monte Carlo method that mathematicians call a *Markov chain*, employs probability to predict sequences of events. A Markov chain, named for the Russian mathematician Andrei Markov, is a sequence of events where the probability for each event only depends on the event just before it. The combination of the two methods is known as *Markov chain Monte Carlo (MCMC)*.

A former student of Adrian Smith’s, David Spiegelhalter, working for the Medical Research Council, a government research funding agency in the United Kingdom, wrote a program for the analysis of complex statistical models using MCMC methods. Spiegelhalter’s program generated random samples using a method called *Gibbs sampling*. He released his BUGS program – an acronym for Bayesian Inference Using Gibbs Sampling – in 1991. BUGS has since become one of the most widely used Bayesian software packages with more than thirty thousand downloads and applications in many different research areas ranging from geology and genetics to sociology and archaeology. Spiegelhalter has applied the Bayesian approach to clinical trials and epidemiology.

The startling growth in Bayesian applications was due both to the availability of manageable numerical methods for estimating posteriors using MCMC sampling and the widespread availability of powerful desktop computers. In our examples, we have only considered simple problems with few variables. In real problems, statisticians typically look to find relationships among large numbers of variables. At the end of the 1980s, there was a breakthrough in applying Bayesian methods. Turing Award recipient Judea Pearl ([B.14.3](#)) showed that *Bayesian networks*, graphical representations of a set of random variables and the probability of two events occurring together, were a powerful tool for performing complex Bayesian analyses. A very simple Bayesian network is shown in [Figure 14.2](#).

As a final example of the advances made by Bayesian analysis, David Heckerman, a machine-learning researcher at Microsoft Research, says, “The whole thing about being a Bayesian is that all probability represents uncertainty and, anytime you see uncertainty, you represent it with probability. And that’s a whole lot bigger than Bayes’ theorem.”⁶ For his PhD thesis at Stanford University, Heckerman introduced Bayesian methods and graphical networks into expert systems to capture the uncertainties of expert knowledge. His “probabilistic expert system” was called Pathfinder and was used to assist medical professionals in diagnosing lymph node disease. At Microsoft



B.14.3. Judea Pearl received the Turing Award in 2011 for developing a calculus for causal reasoning based on Bayesian belief networks. This new approach allowed the probabilistic prediction of future events and also the selection of a sequence of actions to achieve a given goal. His theoretical framework has given strong momentum to the renewed interest in AI among the computer science community.

Research, Heckerman has applied Bayesian techniques to problems such as spam detection in email and troubleshooting failures in computing systems. He now leads a research team analyzing genetic data to better understand the causes of diseases such as HIV/AIDS and diabetes. His team recently performed an examination of the genomes (complete sets of DNA) of many people to find the genetic variants associated with particular diseases, using data from a Wellcome Trust study of the British population. For each of seven major diseases, the data includes genetic information about two thousand individuals with that disease. The data also include similar information for thirteen thousand individuals without any of the diseases. Using a new, computationally efficient algorithm that Heckerman's team has developed to remove false correlations, the researchers analyzed 63,524,915,020 pairs of genetic markers looking for interactions among these markers for bipolar disease, coronary artery disease, hypertension, inflammatory bowel disease (Crohn's disease), rheumatoid arthritis, and type I and type II diabetes. They processed the data from this study using twenty-seven thousand computers in the Microsoft cloud computing platform, an Internet-based service in which large numbers of processors located in a data center can be used on a pay-as-go basis. The computers ran for seventy-two hours and completed one million tasks, the equivalent of approximately 1.9 million computer hours. If the same computation had run on a typical desktop computer, the analysis would have taken twenty-five years to complete. The result was the discovery of new associations between the genome and these diseases, discoveries that could lead to breakthroughs in prevention and treatment.

Computer vision and machine learning: A state-of-the-art application

Humans find vision easy and can look at a scene and rapidly understand the objects in the scene and the context in which these objects coexist. Computer vision still has a long way to go to match human vision even though this has been a key research area in computer science since the mid-1960s. Although progress has been slow, there are now many commercial applications of computer vision algorithms, ranging from industrial inspection systems to license-plate number recognition. In the early 1990s, computer scientists developed vision-based systems to capture three-dimensional human motions. One such system could recover the three-dimensional body positions of a person moving in a special studio wearing clothing with special reflective markers, by collecting images from multiple cameras. Research also continued on algorithms that could recover three-dimensional information from video footage. However, the problems of image understanding and general object recognition remain huge challenges for computer science. Some progress has been made (Fig. 14.3), but for major advances to occur, software models of each object need to be generated from the data rather than handcrafted by the programmer. Machine learning is now recognized to be a key technology for effective object recognition. In 2001, Paul Viola and Michael Jones used machine-learning technologies to build the first object-detection framework that could provide useful detection accuracy for a variety of features. Although their system could be trained to recognize different classes of objects, they were motivated to design their

Fig. 14.3. In the last decade, progress in object recognition in natural images has been achieved with machine-learning algorithms and very large labeled training sets.

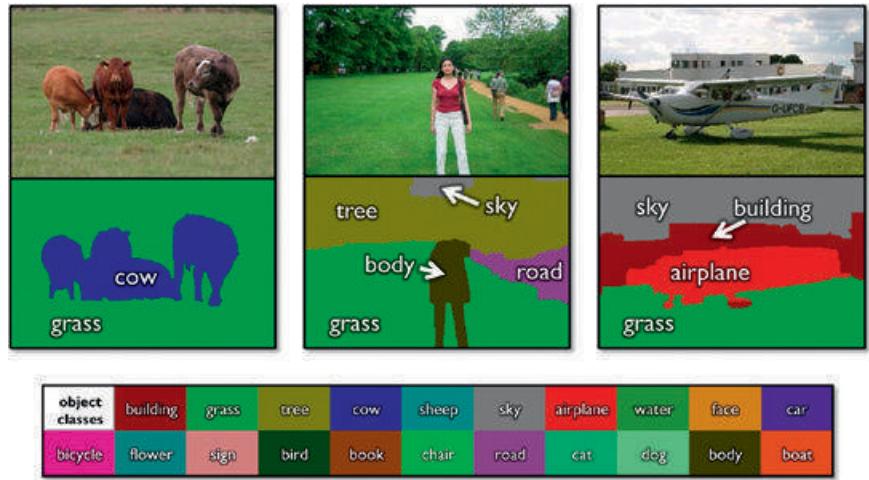
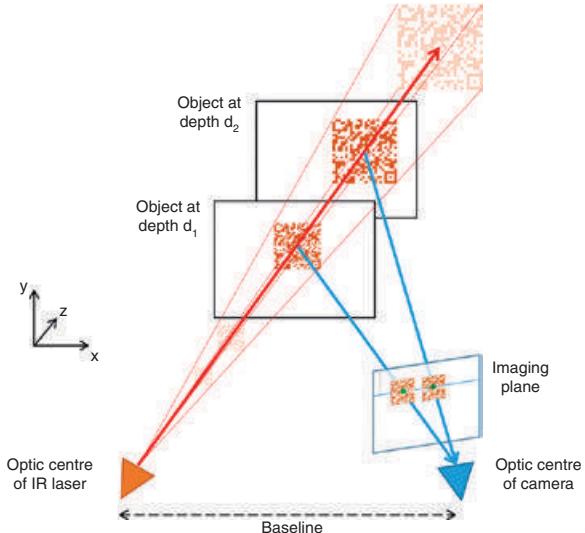


Fig. 14.4. The diagram illustrates the operation of a Kinect 3D Camera. An infrared laser illuminates the scene with a random dot pattern. By using the images of these dots, the camera sensor can determine the relative distance of objects in the scene.

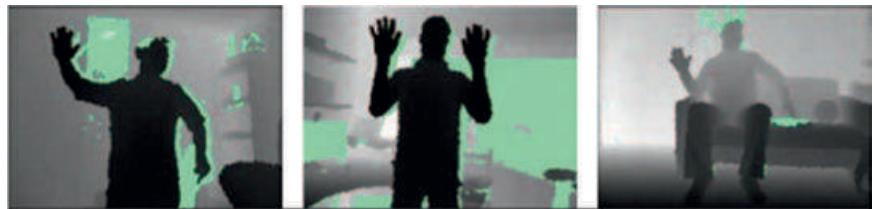


framework to solve the problem of face detection. Their system is now widely used in the face detection software in digital cameras.

In 2008, the team working on the Microsoft Xbox game console met with vision researchers at Microsoft Research's laboratory in Cambridge, England. The Xbox team's ambitious goal was to develop human body-tracking software that was powerful enough to be used for playing computer games without using a game controller. Alex Kipman, from the Xbox team, had taken a new approach to the problem of three-dimensional motion capture by using depth information from an infrared three-dimensional camera. The infrared camera worked at a resolution of 320×240 pixels and generated images at thirty frames per second (Figs. 14.4 and 14.5). Cambridge researcher Jamie Shotton wrote:

The depth accuracy really got me excited – you could even make out the nose and eyes on your face. Having depth information really helps for human pose

Fig. 14.5. Some example images from the Kinect camera. Nearby points are dark gray, and farther points are light gray. In the green areas, no infrared data was captured.



estimation by removing a few big problems. You no longer have to worry about what is in the background since it is just further away. The color and texture of clothing, skin and hair are all normalized away. The size of the person is known, as the depth camera is calibrated in meters.⁷

The Xbox team had built an impressive human body-tracking system using the three-dimensional information but had been unable to make the system powerful enough for realistic game-playing situations. Shotton described the problem as follows:

The Xbox group also came to us with a prototype human tracking algorithm they had developed. It worked by assuming it knew where you were and how fast you were moving at time t , estimating where you were going to be at time $t + 1$, and then refining this prediction by repeatedly comparing a computer graphics model of the human body at the prediction, to the actual observed depth image on the camera and making small adjustments. The results of this system were incredibly impressive: it could smoothly track your movements in real-time, but it had three limitations. First, you had to stand in a particular “T”-pose so it could lock on to you initially. Second, if you moved too unpredictably, it would lose track, and as soon as that happened all bets were off until you returned to the T-pose. In practice this would typically happen every five or ten seconds. Third, it only worked well if you had a similar body size and shape as the programmer who had originally designed it. Unfortunately, these limitations were all show-stoppers for a possible product.⁸

Shotton, with his colleagues Andrew Fitzgibbon and Andrew Blake, brainstormed about how they might solve these problems. The researchers knew that they needed to avoid making the assumption that, given the body position or “pose” in the previous video frame just 1/30 of a second ago, one could find the current body position by trying “nearby” poses. With rapid motions, this assumption just does not work. What was needed was a detection algorithm for a single three-dimensional image that could take the raw depth measurements and convert them into numbers that represented the body pose. However, to include all possible combinations of poses, shapes, and sizes the researchers estimated it would require approximately 10^{13} different images. This number was far too large for any matching process to run in real time on the Xbox hardware. Shotton had the idea that instead of recognizing entire natural objects, his team would create an algorithm that recognized the different body parts, such as “left hand” or “right ankle.” The team designed a pattern of thirty-one different body parts and then used a *decision forest* – a collection of *decision trees* – as a classification technique to predict the probability that a given pixel belonged to a specific part of the body (Fig. 14.6). By



Fig. 14.6. Color-coded pattern of thirty-one different body parts used by the body pose algorithm developed by Microsoft researchers for the Kinect Xbox application.



Fig. 14.7. The 20q game from Radica uses AI technologies to guess the item you are thinking of in twenty questions or less. The game was runner-up for “Game of the Year” in 2005.

predicting these “part probabilities” from a single depth image, they were able to find accurate predictions of the three-dimensional locations of the different joints in the body. The Xbox team was then able to take these predictions and stitch them into a coherent three-dimensional “skeletal” representation of the body.

Decision trees work like the guessing game Twenty Questions (Fig. 14.7), where each question reduces the number of possible answers. For every pixel in the image, the computer asks a series of questions, such as “Is that point on the right of the image more than twelve centimeters farther away than the point under this pixel?” Based on the answer to questions like these, the program proceeds farther down the tree, asking additional questions until it can assign the pixel to a specific body part. The challenge was how best to determine the questions in the tree; the decision trees used in the final system had a depth of around twenty levels and contained nearly a million nodes. The answer was to train the system on a very large set of examples. With the Xbox team, the researchers recorded hours of video footage of actors at a motion capture studio. They filmed the actors performing actions that would be useful for gaming, such as dancing, running, fighting, driving, and so on. These data were then used to automatically animate computer graphic models of different human shapes and sizes. They then simulated the readings that the Kinect sensor would get in a simulation of these actions. The resulting training set contained millions of synthetically generated depth images and the simulated true body positions (Fig. 14.8).

The final challenge was computational. Shotton’s previous work on object recognition in photographs had used training sets of only a few hundred images, and the training phase took less than a day on a single machine (Fig. 14.9). With millions of training images, the Microsoft researchers had to work out how to distribute the training on a cluster of one hundred or so computers. This distributed processing enabled them to keep the training time down to less than a day. With these advances, the researchers and the Xbox team developed very powerful skeletal tracking software and used it to create a whole variety of “controller-free” games. Microsoft launched Xbox Kinect in November 2010 with the marketing slogan “You Are the Controller.” Kinect rapidly became the fastest-selling consumer electronics device in history, according to *Guinness World Records*.

Fig. 14.8. Illustration of three-dimensional image recognition by body parts. The system learns to convert the raw depth images on the left into body part images, and then convert them to a three-dimensional stick version of the body joints.

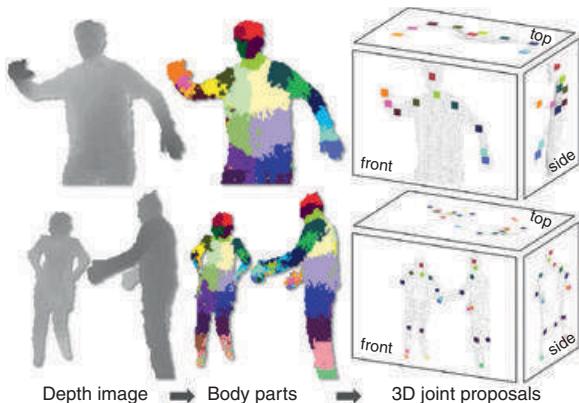
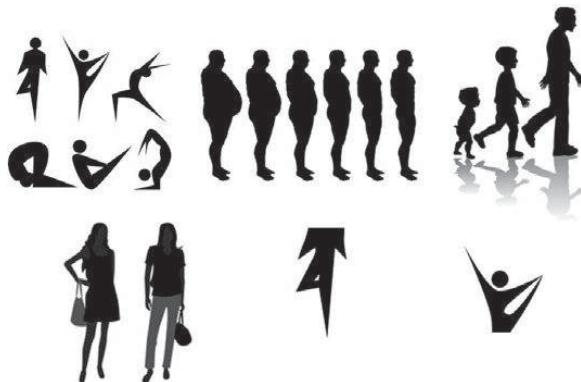


Fig. 14.9. The Kinect tracking system needs to cope with a huge variety of body poses, shapes, and sizes.



Speech and language processing

The idea of using computers to process and understand human language dates back to Turing and Shannon, two of the earliest pioneers of computing. Research into speech and language processing now covers a wide range of sub-disciplines ranging from computational linguistics and natural language processing in computer science, to speech recognition and speech synthesis in electrical engineering. In the 1950s, building on earlier work on language grammars by Shannon, Noam Chomsky (B.14.4) introduced the idea of a “context-free grammar” – a mathematically precise formalism to describe how phrases in a natural language are built from smaller “blocks.” Chomsky’s grammar was able to capture the way in which clauses nest inside other clauses and how adjectives and adverbs are associated to nouns and verbs. Chomsky’s most famous proposal is that knowledge of the formal grammar of a language explains the ability of a human to produce and interpret an infinite number of sentences from a limited set of rules and words. A second research direction in this early period was the development of probabilistic algorithms for speech and language processing. In 1952, researchers at Bell Labs built the first Automatic Speech Recognizer (ASR) system. This was a statistical system that could recognize the first ten digits with 97 percent to 99 percent accuracy – provided the speaker was male, spoke with a 350 millisecond delay between words, and the machine had been tuned to the speaker’s voice profile. Otherwise the accuracy of the system fell to about 60 percent.



B.14.4. Noam Chomsky is a linguist, philosopher, cognitive scientist, and an outspoken defender of democracy and human rights. Since 1955 he has been a professor at MIT. Chomsky is a prolific author who has published more than a hundred books. His idea that children have an innate body of linguistic knowledge – the *Universal Grammar* – has been immensely influential in linguistics. His books about politics, economy, and society have often been controversial.

During the 1960s and 1970s, speech and language processing followed both these directions of research – the formal symbolic approach of Chomsky and the statistical approach of ASR systems. The symbolic approach was typically picked up by the emerging artificial intelligence (AI) community while the statistical approach was mainly followed by electrical engineers and statisticians. Two examples illustrate the different approaches of these two paradigms. As an example application from the symbolic research community, we can take Terry Winograd’s SHRDLU system, written in 1972. This system was able to simulate the behavior of a robot interacting with a world of toy blocks. The program was able to accept sophisticated text commands in natural language such as “Find a block which is taller than the one you are holding and put it into the box.”

Although SHRDLU was a great step forward in natural language understanding, it also showed how difficult it was to build up a computer's understanding of even a very limited world. Meanwhile, the statistical research community was making impressive improvements in speech recognition systems using Hidden Markov Models (HMM). As we have seen, a Markov model is a mathematical system with a set of possible states that undergo random transitions from one state to another, with the new state only depending on the parameters of the previous state. HMMs are systems in which the system being modeled is assumed to have an underlying Markov process but the actual Markov state transitions are not observed directly. The probabilities of the states of the hidden Markov variables have to be deduced from indirect observations. An HMM is one of the simplest Bayesian networks.

By the mid-1990s probabilistic and data-driven models had become the norm for many natural language processing applications. This trend continued in the 2000s with machine-learning techniques delivering results that were clearly superior to those of rule-based systems for almost every aspect of speech and language processing. One important example is machine translation. Instead of developing a system based on a complex set of hand-coded rules, it is now more effective to use large volumes of existing "parallel text" – the same text in both languages – as training data, and have the computer learn how words, phrases, and structures translate in context. Furthermore, with the addition of more human-produced translated text, machine translation systems continue to improve and their quality now exceeds that of the best rule-based systems. With a sufficient corpus of parallel text it is now possible to produce a machine translation system in days rather than the months it would have taken to build a rule-based system. For example, at the time of the Haiti earthquake in 2010, an English-Creole translation system for emergency aid workers was produced in less than five days by the Microsoft Research machine translation team ([Fig. 14.10](#)). It is now possible for endangered language communities to *crowdsource* their own translation system by providing enough parallel text. In this way, Microsoft's Translator Hub service has been used to produce machine translation systems for languages ranging from Hmong and Mayan, to *Star Trek*'s Klingon language, as well as translation systems using vocabularies specific to a particular industry – such as the Russian fashion industry.

The last example of advances in natural language processing concerns speech processing. HMMs enabled us to make great progress in the 1990s. [Figure 14.11](#) shows the reduction in the Word Error Rate (WER) of such systems on a standard benchmark from the U.S. National Institute of Standards and Technology. On the "Switchboard" test data, the WER has fallen from more than 80 percent at the beginning of the 1990s to less than 30 percent by the year 2000. However, for the next decade, despite much research effort by the community, the word accuracy has remained stubbornly the same – until 2009. It was in 2009 that Geoffrey Hinton and colleagues from Toronto and Microsoft Research showed that HMMs that were pretrained using Deep Neural Networks could produce a dramatic reduction in the WER. By 2012 the WER had fallen to less than 10 percent – which is significant, because it means that computer speech processing systems are now approaching human error rates in their accuracy.

Fig. 14.10. The Haiti earthquake in 2010 reduced much of the capital, Port au Prince, to rubble. Using machine learning on the available “parallel” corpus of Creole-English texts – such as the Bible – it was possible to build a translator in less than five days. The National Palace in Port au Prince (a) before and (b) after the earthquake.



IBM's Watson and Jeopardy!

After IBM's success with Deep Blue and computer chess in 1997, the company had been looking for an equally audacious challenge with which to capture the public's imagination (Fig. 14.12). In 2005, after a suggestion from IBM manager Charles Lickel, the director of IBM Research, Paul Horn, tried to interest his researchers in building a machine to beat humans on *Jeopardy!* The game attracted many millions of viewers, and as Horn said, “People associated it with intelligence.”⁹

The American TV quiz show *Jeopardy!* debuted on the NBC network in 1964. It is a quiz game with contestants competing to match questions to answers on a wide variety of topics. The U.S. television host Merv Griffin devised the game. He credits its strange reverse answering style, in which contestants receive clues in the form of answers and must frame their responses as questions, to a conversation with his wife:

My wife Julian just came up with the idea one day when we were in a plane bringing us back to New York from Duluth. I was mulling over game show ideas, when she noted that there had not been a successful “question and answer” game on the air since the quiz show scandals. Why not do a switch, and give the answers to the contestant and let them come up with the question? She fired a couple of answers to me: “5,280” – and the question of

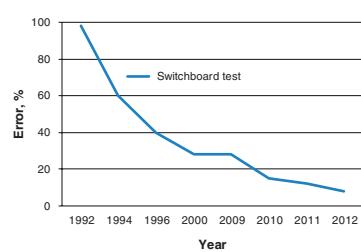


Fig. 14.11. The change in the WER with time for the U.S. National Institute of Standards “Switchboard” test. This shows the dramatic improvement made in the last few years using Deep Neural Network techniques.

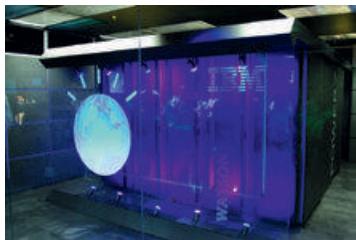


Fig. 14.12. The Watson computer at IBM laboratory in Yorktown Heights, New York, with the Watson logo.

TEN CATEGORIES	MONEY IN BUCKS	JEOPARDY! MONEY IN DOLLARS			
	\$200	\$200	\$200	\$200	\$200
	\$400	\$400	\$400	\$400	\$400
	\$600	\$600	\$600	\$600	\$600
	\$800	\$800	\$800	\$800	\$800
	\$1000	\$1000	\$1000	\$1000	\$1000

Fig. 14.13. A typical *Jeopardy!* game board



B.14.5. David Ferrucci graduated with a degree in biology and a PhD in computer science. His main research areas are natural language processing, knowledge representation, and discovery. He joined IBM in 1995 and led the “Watson/*Jeopardy!*” project from its inception in 2007. After an initial feasibility study, Ferrucci assembled a twenty-five-member team that, in four years, developed a system that not only could “understand” spoken language but also could beat the superstar winners of the question-answering game *Jeopardy!*

course was “How many feet in a mile?” ... I loved the idea, went straight to NBC with the idea, and they bought it without even looking at a pilot show.¹⁰

Each round of the game has six categories, each with five clues for a different amount of money (Fig. 14.13). The categories ranged from standard topics such as history, science, literature, and geography to popular culture and word games, such as puns. An example in the category of “U.S. Presidents” could be the clue “The Father of Our Country; he didn’t really chop down a cherry tree.” The contestant would have to reply “Who is George Washington?” As an example of wordplay, under the category “Military Ranks” could be the clue “Painful punishment practice,” to which the answer is “What is corporal?” The first contestant to “buzz” after the host has read the entire clue wins the chance to have the first guess. With a correct reply, he or she wins the designated amount for the clue; a wrong reply loses the amount of the clue and allows the other contestants a chance to buzz in. The game also has three “Daily Double” clues, which allow contestants to wager a minimum of \$5 up to the maximum of all their winnings, and a “Final Jeopardy!” round where the contestants write down their answer and may gamble all their winnings. If they answer correctly, they win their bet, and a successful gamble can transform the result of the game. The contestant with the highest total after the final round is the winner.

The first *Jeopardy!* superstar contestant was Ken Jennings, a computer programmer from Salt Lake City, Utah. From June until November 2004, Jennings had an amazing seventy-four-game winning streak and won more than 2.5 million dollars. Instead of the audience becoming bored, ratings for the show jumped by more than 50 percent. A key factor in Jennings’s dominance was his lightning fast reflexes: he won the race to the buzzer on more than half the clues.

Paul Horn’s suggestion for IBM to produce a machine to play *Jeopardy!* was controversial. It was not until a year later that he was able to persuade David Ferrucci (B.14.5), head of the Semantic Analysis and Integration Department at IBM Research, to take on the challenge. Ferrucci had many reasons for his skepticism. One of the teams he led was developing a question-answering system called Piquant, short for Practical Intelligent Question Answering Technology. Each year, there was a contest at the Text Retrieval Conference, a gathering of researchers focusing on information retrieval, in which competing teams were given a million documents on which to train their system. In these competitions, based on this very restricted knowledge base, the IBM Piquant system got two out of three questions wrong. In an initial six-month trial period, the Piquant team was trained to answer *Jeopardy!* questions using five hundred specimen clues. Although Piquant did better than a search engine-based approach that used the Web and Wikipedia, it succeeded only 30 percent of the time. From this first disappointing trial, Ferrucci concluded that he needed to adopt a much broader approach that made use of multiple AI technologies. He therefore assembled machine-learning and natural language processing experts from IBM Research and reached out to university researchers at Carnegie Mellon and MIT. Undaunted by the result of the trial, Ferrucci told Horn that he would deliver a *Jeopardy!* machine that could compete with humans within twenty-four months. He gave the project the code name Blue-J. A year later, the resulting machine was christened “Watson” for IBM’s first president, Thomas J. Watson.

In July 2007, Ferrucci and some IBM colleagues flew down to the Sony Pictures Studios in Culver City, California, to meet with Harry Friedman, producer of *Jeopardy!* The result was a provisional go-ahead for a human-machine match in late 2010 or early 2011. Friedman had also agreed that clues with audio or visual clips would not be used. The IBM team now had a deadline to work toward, and they began the process of “educating” Watson. They had access to twenty years of *Jeopardy!* clues from a fan website called J!Archive. From an analysis of twenty thousand clues, the team determined how often particular categories turned up. They could also study individual games, and they analyzed the seventy-four winning games of Jennings’s to understand his strategy. In their “War Room” at IBM Research in Hawthorne, New York, the team plotted this information on a chart they called the “Jennings arc.” He averaged more than 90 percent correct answers, and in one game Jennings won the buzzer on 75 percent of the clues. They calculated that, to beat Jennings, Watson would need to match his precision and win the race to the buzzer at least 50 percent of the time.

One of the early conclusions was that Watson did not need to know literature, music, and TV in great depth to answer the *Jeopardy!* clues. Instead, it needed to know the major facts about famous novels, brief biographies of major composers, and the stars and plotlines of popular TV shows. However, because it could not search the Web during the match, all of this information had to be loaded into Watson’s memory from sources such as Wikipedia, encyclopedias, dictionaries, and newspaper articles, all in a form that the machine could understand.

The biggest obstacle for the researchers was teaching the machine to “understand” what it was supposed to look for from the cryptic *Jeopardy!* clues, which were often worded in a puzzling manner. The first algorithm to be applied was a grammatical analysis identifying nouns, verbs, adjectives, and pronouns. However, there were many possible key words that could be relevant to finding the answer, and Ferrucci and his team had to search through all the many different interpretations. Then, by using a variety of machine-learning methods and cross-checks, they assigned probabilities to a list of possible answers. All of these searches and tests took vital time, and in the game they had to come up with an answer in just a few seconds. Toward the end of 2008, Ferrucci recruited a five-person hardware team to devise a way to speed up the processing time more than a thousand-fold. How was this to be achieved? The answer was to distribute the calculations over more than two thousand processors so that Watson could explore all these paths simultaneously.

During the buildup to the contest, Watson moved on from training on sets of *Jeopardy!* clues to practice matches with previous *Jeopardy!* winners. By May 2010, Watson won against human players 65 percent of the time. The team used Watson’s failures to improve and tune up their algorithms and selection criteria. They also had to insert a “profanity filter” to help Watson distinguish between polite language and profanity. After numerous glitches and many amusing mistakes, Watson’s performance had climbed up the Jennings arc so that it approached the performance of experienced *Jeopardy!* winners. However, the televised match was to pit Watson against two of the very best *Jeopardy!* champions, Jennings and Brad Rutter, who had beaten Jennings in the show’s “Ultimate Tournament of Champions” competition in 2005.

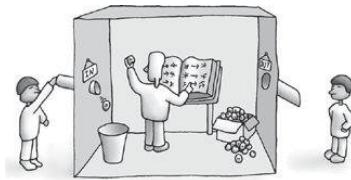


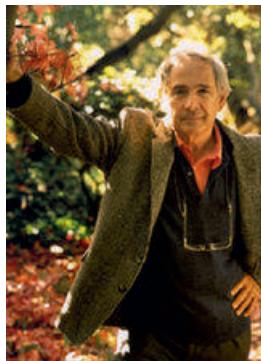
Fig. 14.14. John Searle's "Chinese room" thought experiment showed that a human can follow instructions like a computer and appear to external observers to understand Chinese, without the human having any knowledge of the language.

For the actual game, Friedman and the *Jeopardy!* team insisted that to counter Watson's advantage in electronically "pressing the buzzer," IBM would have to provide Watson with a mechanical finger to physically press a button. In addition, IBM decided that Watson needed a graphical representation for the virtual "face" of Watson. Ferrucci, mindful of the rogue AI computer HAL in Stanley Kubrick's *2001: A Space Odyssey*, suggested, "You probably want to avoid that red-eye look because when it's pulsating, it looks like HAL."¹¹ IBM had just launched its "Smarter Planet" initiative, a campaign to explore how information technology could promote economic growth, sustainable development, and social progress. The icon for the campaign showed planet earth with five bars, representing intelligence, radiating from it, and IBM decided that this image would be the public face of Watson. To make the game more interesting, an answer panel showed the audience – but not the other players, Watson's top five candidate answers, and the confidence the machine assigned to each one. According to Ferrucci, "This gives you a look into Watson's brain."¹²

The match took place at IBM's research center at Yorktown Heights in New York. The producers recorded the show in January and swore the audience and contestants to secrecy until after the match was broadcast in February. The match consisted of two games, and after the first game Watson had the lead. Jennings and Rutter did better in the second game, but Watson won the last Daily Double and thus won the game. In his written answer to the Final Jeopardy! clue, Jennings added a postscript: "I, for one, welcome our new computer overlords."¹³

Watson won the game, but is Watson really intelligent? Ferrucci took the view that "teaching a machine to answer complex questions on a broad range of subjects would represent a notable advance, whatever the method."¹⁴ Prior to the achievements of Deep Blue and Watson, most people would have said that activities like playing chess or competing on *Jeopardy!* required intelligence. However, just as Deep Blue playing chess employed massive computational power to search out the best chess moves, Watson used massively parallel processing to explore and rank a huge number of possible answers to the questions. The IBM Watson team did not try to model the architecture of the human brain, but instead used a host of algorithms for natural language processing and machine learning that gave the machine the ability to (mostly) correctly answer complicated questions. It made no attempt to "understand" the questions as a human would. However, from the point of view of an operational definition of intelligence as in the Turing Test, one might say that both Deep Blue and Watson are intelligent. Most experts, however, would say that Deep Blue and Watson are machines that just simulate intelligence. Such systems, sometimes called *weak AI*, can match human intelligence in a narrow field but not in broader ones. Are such systems a step on the road to *strong AI* – machines that can really think, know, and learn – or are they irrelevant to this goal? This is a question that has generated heated debate among the philosophy and computer science communities.

In a famous thought experiment called the "Chinese Room" (Fig. 14.14), philosopher John Searle argues (B.14.6) against the possibility of strong AI. In the thought experiment, Searle imagines someone who does not know Chinese sitting alone in a room, following directions for stringing together Chinese characters so that people outside the room think that someone inside understands and speaks Chinese. Searle explains:



B.14.6. John Searle is a professor of philosophy at Berkeley in California. Within the computer science community he is best known for his controversial "Chinese room" scenario as an argument against "strong AI."

Imagine a native English speaker who knows no Chinese locked in a room full of boxes of Chinese symbols (a data base) together with a book of instructions for manipulating the symbols (the program). Imagine that people outside the room send in other Chinese symbols which, unknown to the person in the room, are questions in Chinese (the input). And imagine that by following the instructions in the program the man in the room is able to pass out Chinese symbols which are the correct answers to the questions (the output). The program enables the person in the room to pass the Turing Test for understanding Chinese but he does not understand a word of Chinese.¹⁵

Searle first introduced his argument in 1980, and it has generated an enormous number of responses, rebuttals, and counterrebuttals ever since. In a 2011 article, written after Watson's victory on *Jeopardy!* Searle restates his case and concludes, "Watson did not understand the questions, nor its answers, nor that some of its answers were right and some wrong, nor that it was playing a game, nor that it won – because it doesn't understand anything."¹⁶

Key concepts

- Bayesian network
- Bayesian inference
- Posterior beliefs
- Casual reasoning
- Human body tracking
- Universal grammar
- Statistical language translation
- Strong AI
- Chinese room



"It's a rather interesting phenomenon. Every time I press this lever, that post-graduate student breathes a sigh of relief."

15 The end of Moore's law

Will it be possible to remove the heat generated by tens of thousands of components in a single silicon chip?

Gordon Moore¹

Nanotechnology

In 1959, at a meeting of the American Physical Society in Pasadena, California, physicist Richard Feynman set out a vision of the future in a remarkable after-dinner speech titled “There’s Plenty of Room at the Bottom.” The talk had the subtitle “An Invitation to Enter a New Field of Physics,” and it marked the beginning of the field of research that is now known as *nanotechnology*. Nanotechnology is concerned with the manipulation of matter at the scale of nanometers. Atoms are typically a few tenths of a nanometer in size. Feynman emphasizes that such an endeavor does not need new physics:

I am not inventing anti-gravity, which is possible someday only if the laws are not what we think. I am telling you what could be done if the laws *are* what we think; we are not doing it simply because we haven’t yet gotten around to it.²

During his talk, Feynman challenged his audience by offering two \$1,000 prizes: one “to the first guy who makes an operating electric motor which is only 1/64 inch cube,” and the second prize “to the first guy who can take the information on the page of a book and put it on an area 1/25000 smaller.”³ He had to pay out on both prizes – the first less than a year later, to Bill McLellan, an electrical engineer and Caltech alumnus (Fig. 15.1). Feynman knew that McLellan was serious when he brought a microscope with him to show Feynman his miniature motor capable of generating a millionth of a horsepower. Although Feynman paid McLellan the prize money, the motor was a disappointment to him because it did not require any technical advances (Fig. 15.2). He had not made the challenge hard enough. In an updated version of his talk given twenty years later, Feynman speculated that, with modern technology, it should be possible to mass-produce motors that are 1/40 a side smaller than McLellan’s original motor. To produce such micromachines, Feynman envisaged the creation of a chain of “slave” machines, each producing tools and machines at one-fourth of their own scale.



Fig. 15.1. Richard Feynman examining Bill McLellan’s miniature electric motor in 1960. The motor could generate a millionth of a horsepower and Feynman paid McLellan the \$1,000 prize money.

Fig. 15.2. Feynman's letter to McLellan expresses disappointment that McLellan did not need to develop any new technology to build his motor but instead had been able to use tweezers and a microscope.

I am only slightly disappointed that no major new technique needed to be developed to make the motor. I was sure I had it small enough that you couldn't do it directly, but you did. Congratulations!
 Now don't start writing small.
 I don't intend to make good on the other one. Since writing the article I've gotten married and bought a house!
 Sincerely yours,
 Richard P. Feynman



Fig. 15.3. Stanford graduate student Tom Newman wrote the first page of *A Tale of Two Cities* by Charles Dickens using electron beam lithography to form letters only fifty atoms wide.

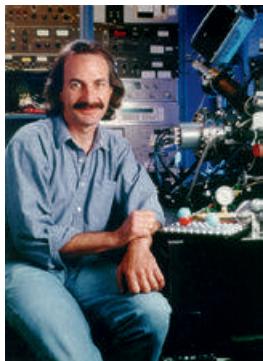
It was not until twenty-six years later, in 1985, that Feynman had to pay out on the second prize. The scale of the challenge is equivalent to writing the entire contents of *Encyclopædia Britannica* on the head of a pin (Fig. 15.3). The winner was Tom Newman, a Stanford graduate student who was using electron beam lithography to engrave patterns on silicon to make integrated circuits. A friend showed Newman a copy of Feynman's 1959 talk and pointed out the section offering a prize for "writing small." Newman calculated he would have to reduce individual letters down to a scale only fifty atoms wide. Using an electron beam machine, he thought it should be possible. To check that the prize was still being offered after all that time, Newman sent a telegram to Feynman. He was surprised to receive a telephone call from Feynman confirming that it was. Because Newman was supposed to be working on his thesis, he had to wait until his thesis adviser went to Washington, D.C., for a few days before he made his attempt. He programmed the machine to write the first page of Charles Dickens's novel *A Tale of Two Cities*. The major difficulty turned out to be actually finding the tiny page on the surface after it had been written. Newman duly received a check from Feynman in November 1985.

Researcher Don Eigler (B.15.1) and his colleagues at the IBM Almaden Research Center in California used the scanning tunneling microscope (STM), invented by their colleagues at IBM Zurich, to manipulate individual atoms and create the world's smallest IBM logo in 1989 (Fig. 15.4). They have also made spectacular quantum "corrals" (Fig. 15.5) and created "artificial" molecules, one atom at a time (Fig. 15.6), confirming another speculation of Feynman's:

It would be, in principle, possible (I think) for a physicist to synthesize any chemical substance that the chemist writes down. Give the orders and the physicist synthesizes it. How? Put the atoms down where the chemist says, and so you make the substance.⁴

In 2012, IBM researchers announced they had used the same technique to store a single bit of information on a magnetic memory made of just twelve atoms. According to researcher Sebastian Loth, it currently takes about a million atoms to store a bit of information on a hard disk. Loth explains that:

Roughly every two years hard drives become denser. The obvious question to ask is how long can we keep going. And the fundamental physical limit is the world of atoms. The approach that we used is to jump to the very end, check if we can store information in one atom, and if not one atom, how many do we need? We kept building larger structures until we emerged out



B.15.1. Don Eigler achieved many breakthroughs in nanotechnology in his laboratory at IBM's Almaden Research Center. His group produced the smallest IBM logo using an STM to position the individual atoms.

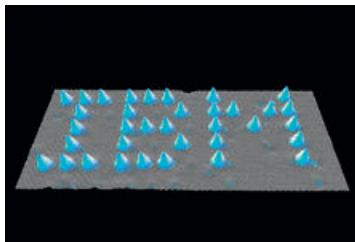


Fig. 15.4. IBM researchers Don Eigler and Erhard Schweizer spelled out the initials of the company in thirty-five individually positioned xenon atoms in 1989.

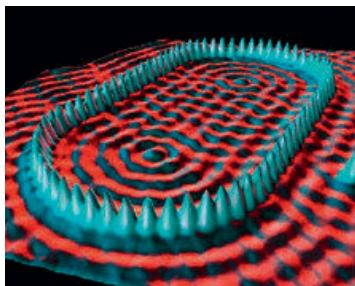


Fig. 15.5. A stadium-shaped “quantum corral” was built by positioning individual iron atoms on a copper surface.

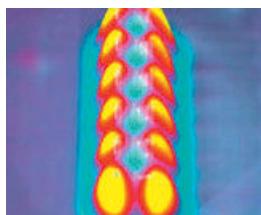


Fig. 15.6. In 2012, IBM researchers built a magnetic memory device consisting of just twelve atoms.

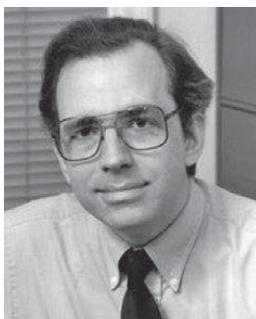
of the quantum mechanical into the classical data storage regime and we reached this limit at 12 atoms.⁵

The groups of atoms were arranged using an STM operating at very low temperatures. By scaling up these twelve-atom bits to a few hundred atoms, it may be possible to make such structures stable at room temperature. Clearly, however, volume production of such memory devices is many years away.

In his 1986 book *Engines of Creation*, the nanotechnology researcher Eric Drexler (B.15.2) envisions a future in which self-replicating nanomachines could be engineered that could create almost any type of matter (Fig. 15.7). In his vision of a nanotechnology-powered future, hunger would be eliminated, all diseases cured, and the human life span extended dramatically. Drexler uses the term grey goo to refer to an out-of-control, spreading mass of self-replicating machines that could literally cause the end of the world. Bill Joy, one of the founders of Sun Microsystems, became so concerned about the potentially catastrophic effects of Drexler’s nanomachines that he warned against unregulated experimentation with nanotechnology in *Wired* magazine. Fortunately, while Drexler’s book certainly excited many people about the future potential of nanotechnology, most scientists believe that we are a long way from actually creating any of Drexler’s self-assembling machines.

The near future

As Gordon Moore acknowledged in 2005 (see Chapter 7), the size of transistors is “approaching the size of atoms which is a fundamental barrier”⁶ for present-day technology. Each year, a group of semiconductor experts in the five leading chip-manufacturing regions in the world – the United States, Japan, Taiwan, South Korea, and Europe – prepare a report called the *International Technology Roadmap for Semiconductors* (ITRS), which identifies the challenges for the future of semiconductor chip manufacturing. In past years, the roadmap has laid out research and development targets necessary for the continuation of *geometrical scaling*, the continued reduction in size predicted by Moore’s law. Now, however, the roadmap also includes *equivalent scaling*, improving performance through innovative design, software solutions, and new materials or structures. The 2012 version of the ITRS looks at both near-term goals, through



B.15.2. Eric Drexler is known for his theoretical work on molecular nanotechnology. He developed the concept of self-assemblers capable of constructing molecules atom by atom. This idea not only captured the imagination of science fiction writers but also created real research interest in this field. There are many skeptics of Drexler’s ideas and the research has not demonstrated the possibility of building nanoscale self-assemblers.



Fig. 15.7. Eric Dexler's vision of nanotechnology included fabricating such things as molecular differential gears.

2018, and long-term goals, 2019 through 2026. On the near-term goal, the ITRS comments:

Scaling planar CMOS [complementary metal oxide silicon, the technology used to build integrated circuits] will face significant challenges. The conventional path of scaling, which was accomplished by reducing the gate dielectric thickness, reducing the gate length, and increasing the channel doping, might no longer meet the application requirements set by performance and power consumption. Introduction of new material systems as well as new device architectures, in addition to continuous process control improvement are needed to break the scaling barriers.⁷

On the longer-term outlook, the ITRS report highlights the problem of managing the power leakage of CMOS devices:

While power consumption is an urgent challenge, its leakage or static component will become a major industry crisis in the long term, threatening the survival of CMOS technology itself, just as bipolar technology was threatened and eventually disposed of decades ago. Leakage power varies exponentially with key process parameters such as gate length, oxide thickness, and threshold voltage. This presents severe challenges in light of both technology scaling and variability. Off-currents in low-power devices increase by a factor of 10 per generation, and will emphasize a combination of drain and gate leakage components. Therefore design technology must be the key contributor to maintain constant or at least manageable static power.⁸

In May 2011, Intel announced the most radical shift in semiconductor technology in fifty years. The new Intel technology uses the latest fabrication process to produce three-dimensional transistors that allow microprocessors to operate faster and use less power than conventional two-dimensional transistors. According to Moore:

For years we have seen limits to how small transistors can get. This change in the basic structure is a truly revolutionary approach, and one that should allow Moore's Law, and the historic pace of innovation, to continue.⁹

Research that led to this breakthrough started in 1997, in a DARPA-funded project at the University of California, Berkeley. The Berkeley team (B.15.3), Chenming Hu, Jeff Bokor, and Tsu-Jae King Liu, looked at the challenge of building a transistor smaller than twenty-five nanometers, ten times smaller than those in production at the time. (A nanometer is a thousand-millionth of a meter.) Two years later, the researchers came up with the idea of a new three-dimensional transistor structure they called a "FinFET" (Fig. 15.8). This is a field effect transistor (FET) formed with a narrow silicon "fin" rising from the surface of the chip. A FET operates by creating an electric field that changes how one of the transistor's semiconductor regions, the gate region, conducts electric current. In a standard two-dimensional FET, the current can only be controlled from the top surface of a silicon channel linking the



B.15.3. The FinFET transistor team at Berkeley. From left to right: Ali Javey, Vivek Subramanian, Ali Niknejad, Jeff Bokor, Chenming Hu, and Tsu-Jae King Liu.

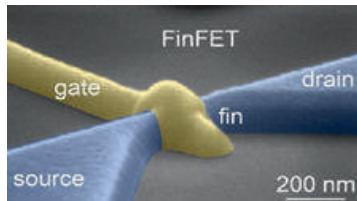


Fig. 15.8. Illustration of a three-dimensional FinFET transistor. Intel began manufacturing twenty-two-nanometer Tri-Gate transistors in 2012.

semiconductor regions. With a fin-shaped silicon channel, the flow of current can be controlled more effectively, using all of the channel's side surfaces. Hu explained the rationale for the fin structure as follows:

An analogy is to think of this channel like a vein. If you want to stop bleeding, you would pinch the vein from both sides. This would be much better than just pressing from one side.¹⁰

In 2000, the Berkeley researchers predicted that FinFET technology could be scaled down to at least ten nanometers, and they estimated that such three-dimensional transistors could move into full-scale production in about ten years. Intel started volume production of its new twenty-two-nanometer, three-dimensional Tri-Gate transistors in 2012 with the announcement of the third-generation Intel Core processor family (formerly code-named Ivy Bridge). The new three-dimensional architecture allows for a 37 percent performance increase at low voltage and a 50 percent power reduction, compared to chips made using conventional two-dimensional technology.

What happens after 2020 or so? Physicist Michio Kaku (B.15.4) has predicted the end of the “Age of Silicon”:

But this process cannot go on forever. At some point, it will be physically impossible to etch transistors in this way that are the size of atoms. You can even calculate roughly when Moore’s law will finally collapse: when you finally hit transistors the size of individual atoms. Around 2020 or soon afterward, Moore’s law will gradually cease to hold true and Silicon Valley may slowly turn into a rust belt unless a replacement technology is found. Transistors will be so small that quantum theory or atomic physics takes over and electrons leak out of the wires. For example, the thinnest layer inside your computer will be about five atoms across. At that point, according to the laws of physics, the quantum theory takes over.... According to the laws of physics, eventually the Age of Silicon will come to a close, as we enter the Post-Silicon Era.¹¹

To see what might happen after 2020, we now take a quick look at three possible postsilicon technologies.

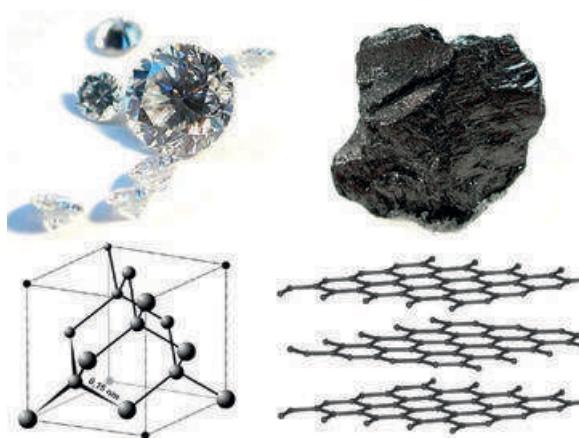


B.15.4. Michio Kaku is an American theoretical physicist and popularizer of science. He has written papers on string theory and several popular science books. He has also hosted several TV programs about science. Kaku predicts that the end of silicon-based computing is near.

A postsilicon age?

The ITRS roadmap is looking toward incorporating nanotechnologies onto a CMOS silicon platform. One of the leading technologies will likely involve new allotropes of carbon. An *allotrope* is a specific structural arrangement of the atoms of an element in crystalline form: for carbon, the two most common allotropes are diamond and graphite. In diamond, each carbon atom uses its four outer electrons to bond with four other carbon atoms to form a tetrahedral structure that is extremely rigid (Fig. 15.9). This structure gives diamond its legendary strength and hardness. For any substance to conduct electric current, it must contain charged particles that can move freely through the material, such as electrons in the outer shell of an atom. In diamond, because all four of the outer electrons in each carbon atom are tied up in bonds between the atoms, the electrons cannot move around freely and so diamond cannot conduct electric current. In graphite, each carbon atom uses only three of its

Fig. 15.9. Carbon takes on different forms depending on how its atoms are arranged. The atoms in a diamond form a rigid pyramid shape. In graphite, the atoms are arranged in flat layers.



four outer electrons to bond to three other carbon atoms, forming flat, parallel layers. Graphite consists of many of these layers of atoms, which can easily slide over each other, making graphite soft. In addition, one of the four outer electrons in each of the carbon atoms in these layers remains free to move and as a result, graphite is a very good conductor of electricity (Fig. 15.9).

Interest in new forms of carbon began more than twenty years ago, when researchers Robert Curl, Harry Kroto, and Richard Smalley (B.15.5) discovered a new, stable form of carbon in which sixty carbon atoms formed a closed spheroidal structure. The carbon atoms were connected in the shape of a soccer ball. Because of its similarity to inventor R. Buckminster Fuller's geodesic dome, the discoverers called this new allotrope of carbon *buckminsterfullerene*, soon shortened to *buckyball* in the popular press (Fig. 15.10). In fact, this carbon 60 allotrope was just the first of a whole new family of hollow carbon structures now known as *fullerenes*, which can take the form of spheres or tubes. In 1991, researcher Sumio Iijima in Japan observed threads of pure carbon that were only about a nanometer in diameter. The walls of these *carbon nanotubes* have the same atomic structures as graphite (B.15.6). The ends of the tubes can either be open or closed. The nanotubes can be up to several centimeters long and have extraordinary strength. IBM researchers have used nanotubes to



B.15.5. The team of Sean O'Brien, Richard Smalley, Robert Curl, Harold Kroto, and Jim Heath that discovered a new stable form of carbon the C_{60} in 1985. Smalley, Curley, and Kroto were awarded the 1996 Nobel Prize for chemistry.

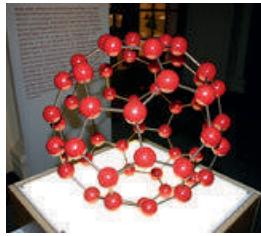
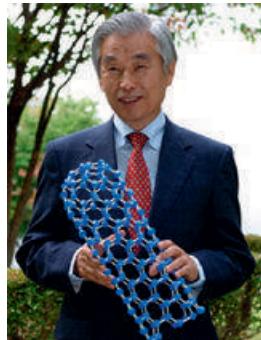


Fig. 15.10. The sixty atom carbon structure discovered by Robert Curl, Harry Kroto, and Richard Smalley. They called this new allotrope of carbon “buckminsterfullerene” in reference to the geodesic domes of American architect and inventor Buckminster Fuller. Inevitably this name is usually shortened to “buckyball.”



B.15.6. Sumio Iijima, discoverer of carbon nanotubes, pictured with a model of a nanotube.



B.15.7. Russian physicists, Andre Geim and Konstantin Novoselov, working at the University of Manchester, England, received the 2010 Nobel Prize in Physics for their discovery of graphene.

make very small, fast transistors. At IBM’s Thomas J. Watson Research Center, researchers have constructed an array of carbon nanotubes on the surface of a silicon wafer and used this silicon to build chips with more than ten thousand working transistors (Fig. 15.11).

In 2004, in Manchester, England, Andre Geim and Konstantin Novoselov (B.15.7) showed how to use graphite to produce a new form of carbon called *graphene*, which consists of an individual sheet of carbon atoms. A single sheet of graphene is just one atom thick and has some remarkable properties. It is the strongest two-dimensional material ever found, able to withstand stress two hundred times greater than steel without tearing apart. In addition, graphene conducts heat better than any metal, and electrons in the two-dimensional layer can move at speeds much faster than in silicon. Geim and Novoselov were awarded the 2010 Nobel Prize in physics “for groundbreaking experiments regarding the two-dimensional material graphene.”¹² Researchers all around the world are looking at all sorts of applications of this new form of carbon – from lightweight, flexible display screens to new types of electronic circuits. In 2010, researchers at IBM used graphene to create transistors that can amplify signals about ten times faster than any silicon transistor.

Our last example of nanotechnology introduces a new type of electronic component. As long ago as 1971, Professor Leon Chua from the University of California, Berkeley, wrote a paper titled “Memristor – The Missing Circuit Element.” In his paper, Chua argued that in addition to the familiar resistor, capacitor, and inductor there was a “missing” two-terminal circuit element. The name *memristor* is derived from *memory resistor*, because the component can change its resistance according to the current flowing but can also “remember” its final state when the voltage is switched off. A memristor is analogous to an unusual sort of pipe whose diameter can expand or shrink according to the amount of water flowing through it. In a memristor, if electrical charges flow in one direction, the resistance of the component will increase, and if charges flow in the opposite direction, the resistance will decrease. If the flow of charges stops, the component remembers the last resistance that it had, and when the flow starts again, the resistance of the circuit will be the same as it was when last active.

Stan Williams (B.15.8) and his colleagues at Hewlett Packard (HP) Labs in Palo Alto, California, have pioneered fabrication of nanoscale memristors (Fig. 15.12). The devices have advantages over conventional silicon-based memory in terms of access speed, power, and density, and can be fabricated using conventional silicon lithography techniques. HP’s process to create an array of memristors consists of laying down a set of parallel “nanowires” – less than about ten nanometers wide – coated with a layer of titanium dioxide a few nanometers thick. A second set of wires is then laid down at right angles to the first set, and the crossover points of these wires are the memristors. Commercial versions of memristor memory chips will likely appear in the next few years, but it will be some time before such technologies present a significant challenge to *flash memory*, the durable, rewriteable memory chips used in digital cameras, smart phones, and other portable devices.

Fig. 15.11. An illustration of a carbon nanotube transistor. IBM has built chips with more than ten thousand nanotube transistors.

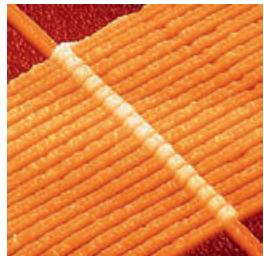
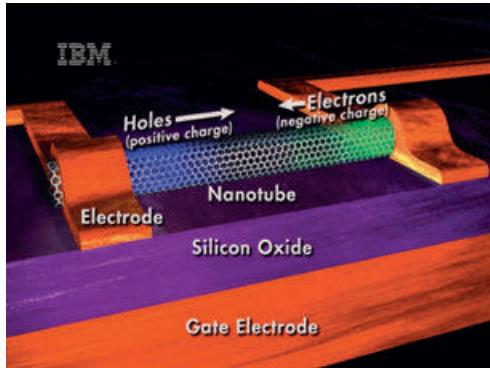


Fig. 15.12. New electronic components called *memristors* have the potential to transform the market for solid-state memory devices. A memristor has resistance to electrical current, but the resistance changes as the current changes. When the current is removed, the memristor preserves the memory of its last resistance. In this image, each of the white spots is a memristor only fifty nanometers in diameter.

Quantum computing

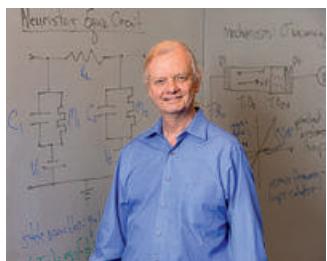
The study of the limits imposed by quantum mechanics on computers probably became respectable as an academic field after physicist Richard Feynman gave a keynote talk at a conference on the “Physics of Computation” at MIT in 1981. In his speech, Feynman talked about the problem of performing a computer simulation of physics:

I'm not happy with all the analyses that go with just the classical theory, because Nature isn't classical, dammit, and if you want to make a simulation of Nature, you'd better make it quantum mechanical, and by golly it's a wonderful problem, because it doesn't look so easy.¹³

Feynman proposed building a computer out of elements that obey quantum mechanical laws:

Can you do it [simulate quantum mechanics] with a new kind of computer – a quantum computer? ... It's not a Turing machine, but a machine of a different kind.¹⁴

As we have seen, the basic principles of a Turing machine – that simple, theoretical computational device devised by Alan Turing in 1936 – underlie the operation of all conventional computers. Yet, as Feynman pointed out, a computer operating according to the laws of quantum mechanics would be a new kind



B.15.8. Stan Williams received a doctorate in physical chemistry from Berkeley. He is director of the Memristor Research Group at HP Labs in Palo Alto. In 2000, Williams was awarded the Feynman Prize in Nanotechnology.



B.15.9. David Deutsch developed the theory of a universal quantum Turing machine. In a famous paper published in 1985, he argued that if a quantum computer could be built, it would have some remarkable properties due to quantum parallelism.

of computer, one that might be able to do calculations that conventional computers cannot do. Feynman was referring specifically to simulations of quantum systems to calculate quantum wave functions and quantum probabilities. After Feynman's lecture, David Deutsch (B.15.9), a physicist at the University of Oxford, took the next step. In 1985, Deutsch proved that a quantum computer could indeed do some calculations faster than a conventional computer. But it was not until 1994 that interest in quantum computing really exploded when Peter Shor (B.15.10) of Bell Laboratories discovered a *quantum algorithm* that could potentially solve certain mathematical problems much faster than the best algorithm running on a conventional computer.

What are the key elements of a quantum computer? First, we are only allowed to use quantum objects, like electrons or atoms, to input and store information, and to perform logical operations on this information. Quantum algorithms are executed using these fundamental logical operations. Finally, we need to be able to read out the answer to our quantum calculation. In his talk in 1981, Feynman speculated about the possibility of storing a single bit of information using the quantum states of a single electron. As we discussed in Chapter 7, electrons possess a property called *spin*. In quantum mechanics, an electron can exist in one of two possible spin states, which we call *spin up* \uparrow and *spin down* \downarrow . To represent digital information, we can use the spin up state \uparrow to represent a 1 and the spin down state \downarrow to represent a 0. But this is not the whole story: in quantum mechanics, the electron can be in a *quantum superposition* of both these states. The electron's state is described by a *probability amplitude*. Using the traditional symbol ψ to represent the probability amplitude, this quantum superposition can be written:

$$\Psi = \alpha \uparrow + \beta \downarrow$$

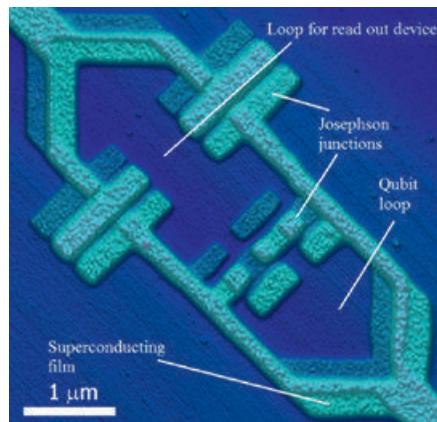


B.15.10. Peter Shor received a PhD in applied mathematics from MIT in 1985. While working at Bell Labs he became famous for his quantum factorization algorithm that he discovered in 1994. Shor has been a professor at MIT since 2003.

where α and β are the amplitudes of the two possible spin states. What happens if we make a measurement of the spin of an electron in such a quantum superposition? According to standard quantum mechanics, we must observe the electron in either a spin up state or a spin down state, but for any given electron in the state Ψ it is impossible to predict with certainty which spin state we will see. However, if we were to prepare an *ensemble* collection of many different electrons in exactly the same way, so that each of them is in the same state Ψ , then quantum mechanics does make a definite prediction. If we make measurements of the spin state of all of the electrons in this collection, quantum mechanics predicts that we will obtain the spin up result with probability α^2 and the spin down result with probability β^2 . The total probability to get any of the possible results must always add up to one, so the sum of these two probabilities must add up to one.

In some sense, we can say that the electron in superposition state Ψ is in both spin states at the same time. So now we see that if we use an electron to represent digital information, in addition to being in one of the 1 and 0 states, the electron could also be in a superposition of both the 1 and 0 states with probabilities determined by α and β . After more than half a century studying the fundamentals of computation, physicists had discovered something new about information at the quantum level. Information stored in a quantum

Fig. 15.13. A superconducting Josephson Junction qubit device made by researchers at Delft University of Technology.



system therefore requires a new name, a *quantum bit* or *qubit* (Fig. 15.13). This superposition property of quantum states is one of the two key properties of quantum mechanics that give quantum computers their remarkable power. In a conventional computer, a bit can have a value of either 0 or 1. In a quantum computer, a qubit can also be in a quantum superposition and so can be both 0 and 1 at the same time. A system with two qubits can hold four values simultaneously – 00, 01, 10, and 11.

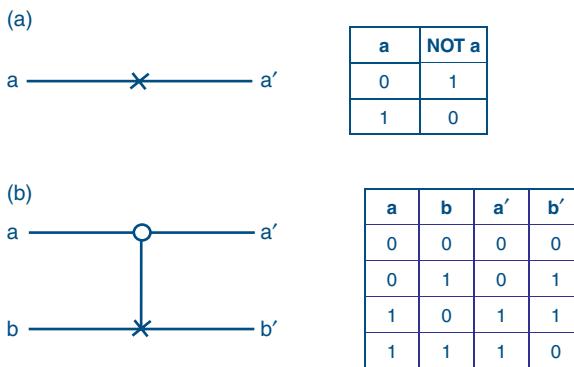
When MIT computer scientist Edward Fredkin (B.15.11) visited Feynman at Caltech in 1974, Fredkin was researching the seemingly strange problem of how to build a *reversible* computer. This is a type of computer that would be able to reverse calculations – “uncalculating” – as well as being able to calculate forward in the usual way. In conventional computers, logical operations are performed by logic gates implemented in silicon. The familiar “AND” gate is shown in Figure 2.8 with its two inputs and one output. All the possible inputs and outputs for an AND gate are summarized in the accompanying truth table. From this truth table, we see that an AND gate outputs a 1 only if both its inputs are 1; for the other three possible input combinations, the gate outputs a 0. The AND gate is therefore not reversible in the sense that it is impossible to deduce a unique input signal from just the output signal. Fredkin devised a new set of logic gates that are reversible – that is, gates such that the output signal from the gate uniquely determines the input signal. The simplest example of one of Fredkin’s gates is the “Controlled NOT” or CNOT gate. This gate is shown in Figure 15.14 together with a conventional NOT gate and the corresponding truth tables. From the truth table for the CNOT gate, we see that the bottom input either “does nothing” or acts as a conventional NOT gate, reversing a 1 to a 0 and vice versa. Which action is chosen is determined by the signal on the upper input, which acts as a control line. If the upper input is a 0, the lower line does nothing. If it is a 1, the lower line acts as a NOT gate. Fredkin showed that it was possible to perform every logical operation using a complete set of such reversible gates (more than just the CNOT gate).

Why do we need to bother about reversible gates? Such gates are relevant for quantum computing because the laws of quantum mechanics are reversible in time. Reversibility is a property of conventional physical waves, not just



B.15.11. At the age of nineteen, Ed Fredkin left Caltech and joined the U.S. Air Force to serve as a fighter pilot. He became a professor at MIT in 1968 and was director of project MAC from 1971 to 1974. He was a close friend of Richard Feynman’s and introduced him to the concept of reversible computing. Fredkin’s research interests are wide-ranging and include the physics of computation and cellular automata.

Fig. 15.14. Edward Fredkin devised a set of logic gates that are reversible in the sense that the output signal from the gate fully determines the input signal. (a) A classical NOT gate and its truth table, and (b) a controlled NOT or CNOT gate and its truth table.



these strange quantum probability waves. A wave traveling in one direction along a string, for example, can just as easily travel in the reverse direction. This reversibility property of quantum mechanics means that if we wish to construct a quantum computer, we have to use computational elements that are reversible.

We can now write down the essential ingredients of a quantum computer. There must be a physical system in which information can be stored as qubits on individual quantum objects such as electrons, atoms, or photons. The information can be not only the familiar digital 1s and 0s but also quantum superpositions of 1 and 0. A quantum computer must have mechanisms by which these qubits can be made to interact so that we can perform Fredkin's reversible logic operations. Note that because we could choose to start off our quantum computer in a quantum superposition of all the possible initial states, in principle the quantum computer would calculate results for all the possible logical paths at the same time. David Deutsch, who first proved that quantum computers can be more powerful than conventional computers, called this property *quantum parallelism*. But how to exploit this property is not so obvious. According to standard quantum theory, making a measurement on a quantum superposition will result in only one of the possible states being selected, so how can quantum parallelism actually be useful? Shor's great contribution was to find a way to extract just a little information from all these quantum paths.

There is a second key feature of quantum mechanics that we must now explain, called *quantum entanglement*. Entanglement is a feature of certain types of two-particle quantum states that we can think of as having some invisible wiring to share information between the two particles (Fig. 15.15). We can illustrate the strange nature of entanglement by considering a thought experiment from particle physics. There is an unstable particle called a *neutral pion* that most of the time spontaneously decays into two photons (a photon being a particle-like bundle of light energy). On some occasions, however, the pion decays into an electron (e^-) and its antiparticle, a positron (e^+), instead of two photons. This is a rare occurrence for the pion, but it gives us the simplest experiment to illustrate what is meant by quantum entanglement. As in classical physics, *angular momentum* must be conserved in any quantum mechanical process. The

Fig. 15.15. Experiments with quantum entanglement were carried out using optical fibers running under Lake Geneva in Switzerland by Nicolas Gisin from the University of Geneva.



pion has zero spin, and because angular momentum must be the same before and after the decay, the spins of the electron-positron pair must be in opposite directions for conservation of angular momentum. If we also start with the pion sitting at rest, conservation of linear momentum dictates that the electron and positron must fly off in opposite directions (Fig. 15.16). If we just focus on the spin state of the two particles, there is probability $\frac{1}{2}$ for the positron to be in the spin up state \uparrow with the electron going in the opposite direction in the spin down state \downarrow . Similarly there is probability $\frac{1}{2}$ for the positron to be in the spin down state \downarrow and the electron in the spin up state \uparrow . What this means is that if we measure the spin of the positron to be spin up \uparrow even though the particles may be widely separated in space, we know instantly that the spin of the electron traveling in the opposite direction must be spin down \downarrow . Similarly, if the positron is measured to be spin down \downarrow , we know instantly that the electron is spin up \uparrow . The spin information is shared – “entangled” – between the two particles.

It was the physicist Erwin Schrödinger who came up with the wave equation that determines how quantum probability waves evolve with time. Schrödinger was familiar with superposition and the physics of waves from classical physics. From the earliest days of quantum mechanics, he used the term *entangled* to describe such two-particle states and said of this entanglement property:

I would not call that *one* but rather *the* characteristic trait of quantum mechanics, the one that enforces its entire departure from classical lines of thought. By the interaction the two representatives (or ψ -functions) have become entangled.¹⁵

We can now do experiments to verify these spin measurement predictions in situations where the information about the measurement of the first spin could not have influenced the second measurement on its separated partner – unless the information traveled faster than the speed of light. Albert Einstein

greatly disliked what he called the “spooky action at a distance”¹⁶ effect needed to explain these surprising quantum spin correlations.

Because entanglement is entirely nonclassical, it may not be surprising that a quantum computer acting on entangled states can lead to results beyond the power of a classical computer. We can easily see how such entangled states can arise in quantum computation. Consider the action of a quantum CNOT gate on a two-qubit state (see box on Quantum Entanglement). When the two-qubit states are just simple products of single-particle 1 and 0 states, we obtain the exact analog of the classical result. But if one of the qubits is in a superposition state of 1 and 0, acting on this state with a quantum CNOT gate yields an entangled two-qubit state just like the example of the pion decaying into an electron and positron. It is this nonclassical feature of quantum mechanics that gives quantum computers their extraordinary properties.

Quantum entanglement

For a pion decaying at rest to a positron-electron pair (e^+e^-), the positron and the electron move away from each other in opposite directions as shown in Fig. 15.16 (a). Since the pion has zero spin, the net spin of the positron-electron pair must also be zero because of conservation of angular momentum. However, the spin state of either the positron or the electron is not definitely known and the spin state of the pair is said to be entangled. The entangled wave function for the pair is shown in Fig. 15.16 (b). If we measure the positron spin to be \uparrow_{e^+} then we know immediately that the spin of the electron must be \downarrow_{e^-} and vice versa. Since the positron and the electron are moving apart, this quantum correlation of spin measurements can be over long distances.

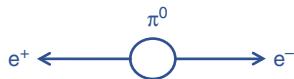


Fig. 15.16. Addition - Pion decay:
 $\pi^0 \rightarrow e^+e^-$

a) Pion decaying at rest to a positron-electron pair (e^+e^-).

$$\Psi_{e^+e^-} \sim \left(\uparrow_{e^+} \downarrow_{e^-} - \downarrow_{e^+} \uparrow_{e^-} \right)$$

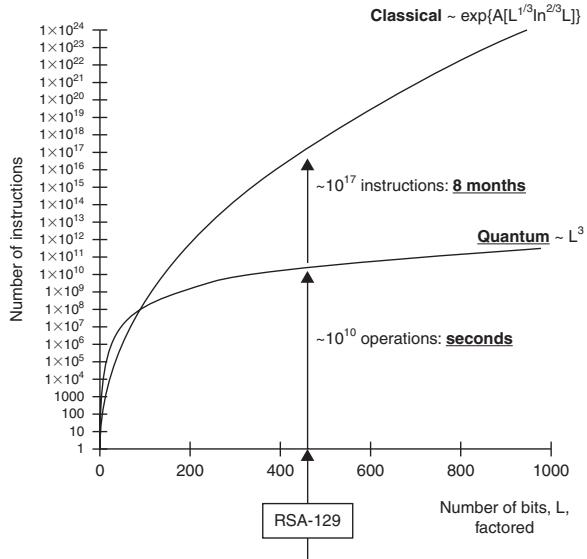
b) Entangled spin state of the positron-electron pair resulting from the pion decay.

Quantum computation often involves entangled states. These can arise from the action of a quantum CNOT gate on a two qubit state. The equations a, b, and c below show the action of a CNOT gate on three different two qubit states. With qubit input 1 on the upper control line of the gate, a qubit input 0 to the bottom line is flipped to a 1 as shown in (a). With qubit input 0 to the upper control line, a qubit input 0 to the bottom line is left unchanged as in (b). However, if the qubit input on the control line is a superposition ($1 + 0$) the action of the CNOT gate on a 0 qubit input to the bottom line produces the entangled state shown in (c).

For cases (a) and (b) the action is straightforward and each particle is in a definite spin state before and after the CNOT gate. Acting with a CNOT gate on a two qubit superposition input state on the upper control line produces an entangled state in which neither particle is in a definite spin state as shown in (c).

- (a) $1_1 0_2 \xrightarrow{\text{CNOT}} 1_1 1_2$
- (b) $0_1 0_2 \xrightarrow{\text{CNOT}} 0_1 0_2$
- (c) $(1_1 + 0_1) 0_2 \xrightarrow{\text{CNOT}} (1_1 1_2 + 0_1 0_2)$

Fig. 15.17. Factorizing RSA-129. This graph shows the increase in computing power, measured in numbers of computer instructions, required to factorize larger and larger numbers, measured in numbers of bits. For a classical computer, the required power grows exponentially with the number of bits in the number to be factorized. The importance of Peter Shor's quantum algorithm was that it showed that with a quantum computer, the required power grows only as the cube of the number of bits. Also shown is the 129-digit number RSA-129 that was factorized in 1994 by volunteers using about 1,600 computers over several months. A quantum computer operating at the same speed as just one of these machines could factorize the number in only a few seconds.



Conventional computers are very good at multiplying two numbers together. For example, the time taken to multiply two N digit numbers grows as the square of N . By contrast, the time needed to factorize an N -digit number – that is, to resolve the number into two smaller numbers that when multiplied together form the larger number – grows faster than any power of N . This is an example of a *one-way function*, as explained in our discussion of public-key cryptography in Chapter 12. A one-way function is a mathematical problem that is easy to solve in one direction, but difficult or even impossible to solve in the other. For example, it is easy to multiply together two large *prime numbers* (numbers divisible only by themselves and 1). However, if you give the huge number resulting from that multiplication to someone else and ask him or her to tell you what numbers you started with, this problem is very hard. Shor showed that a quantum computer could, in principle, factorize numbers just as easily as it multiplied them, without the computing time increasing unreasonably as the size of the number to be factorized grows. This ability is astonishingly powerful. As we have seen, the whole basis of the RSA cryptosystem – named for its inventors, the computer scientists Ronald Rivest, Adi Shamir, and Leonard Adleman – is the computational difficulty of factorizing large numbers. For example, in 1994, the 129-digit number known as RSA-129 required eight months to factorize, using more than 1,600 computers (Fig. 15.17). If we could build a quantum computer that was roughly the same speed as just one of the computers used in this trial, Shor's algorithm could factorize RSA-129 in less than ten seconds. For this reason alone, many government agencies around the world are now funding attempts to build a quantum computer.

The computer scientist Lov Grover discovered another interesting class of algorithms in 1997. Grover's quantum search algorithm showed that a quantum computer could greatly increase the speed of searching a database. An example would be trying to find the name of a person in a telephone directory if you only know their telephone number. For a database with N items, Grover's

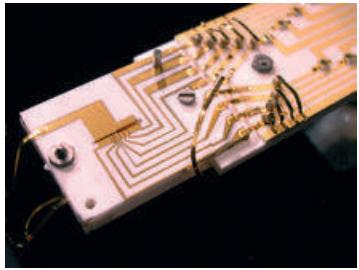


Fig. 15.18. Physicists at NIST in Boulder, Colorado, have demonstrated sustained, reliable quantum information processing in the ion trap at the left center of this photograph. The ions are trapped inside the dark slit – 3.5 millimeters long and 200 microns wide – between the gold-covered alumina wafers. By changing the voltages applied to each of the gold electrodes, scientists can move the ions between six zones of the trap.

algorithm reduces the number of steps needed to find the answer from N to the square root of N . So, for a database with a million entries, a quantum computer could find the correct entry in only one thousand steps.

So how much progress has been made toward actually building a quantum computer? It is a fast-moving field, and many groups around the world are exploring different ways to store and manipulate qubits. In 1995 Ignacio Cirac and Peter Zoller from the University of Innsbruck showed how the energy levels of trapped ions could be used to store qubits and how a quantum CNOT gate could operate on these qubits. In ion traps, ions (electrically charged atoms) are confined by an arrangement of electric fields so that the ions are kept suspended in space. The whole system needs to be in an almost complete vacuum, and the ions must be cooled to near absolute zero to remove their vibrational energy. The ions then arrange themselves in a linear array. After Cirac and Zoller's paper, Nobel Prize recipient David Wineland's (B.15.12) team at the National Institute of Standards and Technology (NIST) became the first to demonstrate quantum logic operations on qubits stored on trapped ions. Two energy levels of the ion are used as the qubit states, which are prepared and measured by directing laser beams at specific ions. Coupling between the ions is provided by the vibrational states of the ions in the ion trap. Using these techniques, the researchers were able to isolate systems containing a few qubits and to construct a quantum gate. More recently, Wineland's group stored qubits using two beryllium ions that can be moved between different zones of the ion trap by applying electric fields (Fig. 15.18). They were able to initialize and store the qubits on the ions in any desired starting state and then perform logic operations on the qubits. They were also able to transfer quantum information between the different zones in the trap. Using these techniques, Wineland's team successfully performed a sequence of four single-qubit operations, one two-qubit operation, and ten transport operations (Fig. 15.19). To scale beyond ten to one hundred trapped ion qubits, Wineland and his group have proposed using what they call a *quantum charge coupled device* (QCCD) (Fig. 15.20). Its operation will require very precise control of the ion positions as they are shuttled from region to region. Wineland's team notes that “scaling to thousands or more qubits in the QCCD may be challenging.”¹⁷

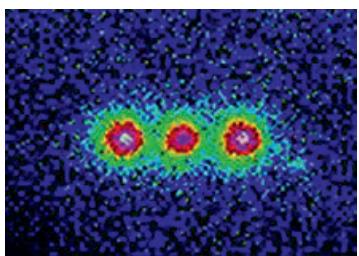


Fig. 15.19. David Wineland's research group at NIST designed and built this trap to confine three magnesium ions. A team of researchers in Innsbruck, Austria, have now been able to store an array of fourteen entangled qubits in an ion trap.

Although operation of such complex ion-trap systems is still very delicate, ion-trap technology does allow the use of simple quantum algorithms. But how close are we to creating a quantum computer that could factorize a number with hundreds of qubits? To factorize RSA-129 with 426 bits, we would need to build a quantum computer with close to a thousand qubits of memory that can execute about a billion quantum gate operations. There are other problems for would-be builders of quantum computers. Conventional computer memories suffer from the problem that individual bits can occasionally get “flipped.” Cosmic rays, for example, are one cause of such errors. To counter this problem, the computer industry has developed a wide range of error detection and correction techniques. A simple example is a *parity check* in which the 1s and 0s are added before and after sending a message. If a 1 has been corrupted to a 0, or vice versa, a parity check will reveal the error. Computer engineers have devised more complicated techniques to handle situations where more than one error has occurred and also ways of detecting which bit has flipped and then correcting it. For qubits, we have all these problems and more. Not only

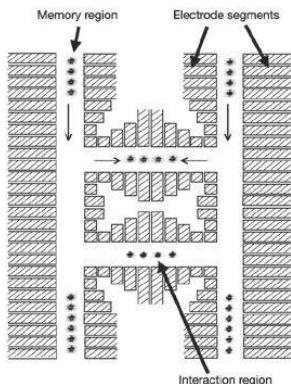
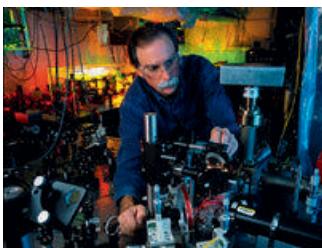


Fig. 15.20. David Wineland and his team at NIST propose using a quantum charge-coupled device for scaling up ion trap qubits.



B.15.12. Nobel Prize recipient David Wineland adjusts a laser beam used to manipulate ions in a low-temperature, high-vacuum ion trap. Wineland's group at the NIST laboratory in Boulder, Colorado, have demonstrated all the key elements required to build a quantum computer.

can we have random bit flips, but also the phase relationship between the different states in a quantum superposition can be affected by interactions with the surrounding environment. Surprisingly, it turns out to be possible in principle to detect and correct such quantum errors. Andrew Steane from Oxford and Peter Shor at Bell Labs independently devised schemes that use quantum entanglement to protect and correct quantum data. Such quantum error-correction techniques will require an order of magnitude more qubits, and it remains to be seen whether such methods will be feasible in practice.

Ion traps are only one technology that researchers are investigating for developing a quantum computer. Several research groups are investigating building qubit systems using a *Josephson junction*, an insulating barrier separating two superconducting materials. Superconducting electron pairs can travel through the barrier by tunneling. Other groups are working to manipulate spins on electrons bound to atoms embedded in a silicon chip. Another exciting approach is exploring the possibility of *topological quantum computing*. The Russian-born physicist Alexei Kitaev first suggested looking for quantum systems with a *topological excitation*. We can illustrate the idea of a topological excitation in terms of the vibrations of an elastic band stretched between two points and anchored at both ends. A topological excitation is analogous to vibrations of a band with a twist. There is no way for the band to untwist itself. Similarly, information stored in a topological qubit would be automatically protected against errors caused by interactions with its surroundings, which eliminates the need for quantum error correction. Research into such solid-state systems is still at an early stage but topological quantum computers may be the best bet to deliver quantum computers that can handle large numbers of qubits.

Synthetic biology and DNA computing

The intersection of computing, nanotechnology, and biology is an exciting area of research (**B.15.13** and **B.15.14**). The research field of *synthetic biology* is attempting to produce standard biological components using the principles of computer science and engineering. Tom Knight (**B.15.15**), an MIT researcher who studies the intersection between computing and biology, says:



B.15.13. Nadrian "Ned" Seeman is a professor at New York University and one of the founders of structural DNA nanotechnology. He studied biochemistry and crystallography and since the 1980s he has been researching the structural properties of DNA molecules. In 1991 he managed to construct a cube from DNA molecules by orienting them in an electric field. In 1995 he was awarded the Feynman Prize in Nanotechnology. Together with Don Eigler from IBM he won the Kavli Prize in Nanoscience in 2010 "for their development of unprecedented methods to control matter on the nanoscale."^{B1}



B.15.14. Randy Rettberg studied electrical engineering and computing. In the 1990s as a major career change, he decided to quit his job with Sun Microsystems and apply his engineering knowledge to molecular biology. Rettberg is president of the International Genetically Engineered Machine Foundation. The organization runs a global competition for undergraduates and high school students in designing brand new biological parts for “genetically engineered machines.”

The key ideas of modern engineering – modularity, modeling, hierarchical design, isolation of concerns, abstraction, reusable parts, defined interfaces, design rules, flexibility – promise to be just as applicable to biological systems as they are to computers or aircraft....

But the real challenge is learning to engineer with unique characteristics of biological systems: their self-reproducing capability, their evolutionary capacity to adapt, and their remarkable robustness in the face of damage and imperfect or failing components. These organizational engineering principles will play an important role not just in engineering biological systems, but in engineering in our existing disciplines.¹⁸

One key goal of synthetic biology is to produce a catalog of standard biological devices that biological engineers can put together to design new life systems. We will look at another avenue of research based on DNA sequences.

Engineering with DNA is an extreme example of molecular nanotechnology. Humans have about one hundred trillion cells, and most human cells are between one and one hundred micrometers in diameter. Each cell contains a nucleus, where most of our genetic material is stored as DNA. Inside the nucleus, the DNA is organized in linear molecules called *chromosomes*. The genetic information in DNA is stored as a code consisting of four nitrogen-containing compounds called *bases*: adenine (A), guanine (G), cytosine (C), and thymine (T). Sequences of these bases determine the genetic instructions for maintaining and replicating cells. Because every base must be one of these four types, each base encodes two bits of information. There are about 3.5 billion bases in human DNA, so the entire human genome, a complete set of all our genetic instructions, corresponds to about seven billion bits of information or less than a gigabyte to store the whole human genetic code. The bases pair up in a specific way with each other – A with T and C with G – to form what are known as *base pairs*. The DNA molecule is shaped like a twisted ladder, a structure called a double helix. Each rung of the ladder consists of a base pair. The base pairs are attached to the sides of the ladder, which consist of sugar and phosphate molecules (Fig. 15.21). This is the famous double helix of Francis Crick and James Watson.

An important property of DNA is that it can be exactly copied so that the cell can divide into two new cells, each with an exact copy of the original DNA. The fundamental unit of heredity for individuals is a gene, a sequence of DNA that provides instructions for making a specific protein. These gene sequences range from a few hundred to more than two million base pairs in length, and



B.15.15. Tom Knight studied electrical engineering at MIT and worked on the ARPANET in the 1960s and 1970s. He was a graduate student in the Artificial Intelligence Lab at MIT and received a doctorate in integrated circuit design in 1983. In the 1990s he became interested in biology and started working with simple bacteria called mycoplasmas. By modifying the DNA, Knight managed to assemble a synthetic bacterial cell. From his research he developed the concept of BioBricks – standard sections of DNA that can be joined together in different ways to create organisms that can perform some specific functions. There are now more than ten thousand parts in the BioBricks registry.

Fig. 15.21. Illustration of the double helix of the DNA molecule.

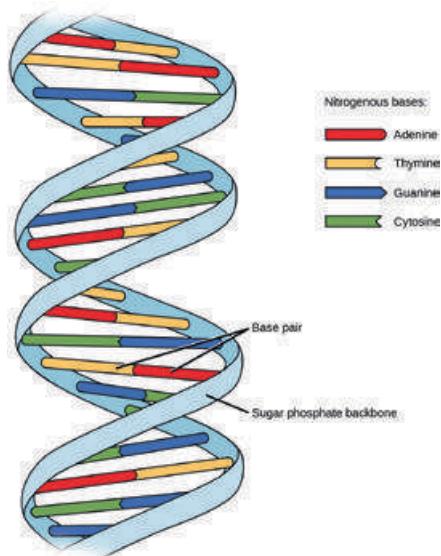


Fig. 15.22. Researchers at the University of California, San Diego, genetically engineered bacteria to glow and blink in unison, acting as a tiny “neon” sign.

a chromosome contains many such genes. The Human Genome Project estimated that humans have between twenty thousand and twenty-five thousand genes stored in twenty-three pairs of chromosomes. Thus a cell stores a gigabyte of genetic information in a volume as small as about a millionth of a cubic millimeter (Fig. 15.22).

Understanding the genetic basis of cells means that it is possible in principle to manufacture a DNA sequence to order and produce a synthetic version of a cell’s genome. In 2010, researchers at the J. Craig Venter Institute in Rockville, Maryland, announced the creation of the first synthetic life form, a self-replicating bacterial cell (Fig. 15.23). The team synthesized the genome of an existing bacterium consisting of more than one million base pairs and inserted this synthetic genome into a different bacterium with its own genome removed. The new genome took over the cell’s machinery, changing the cell’s appearance and behavior, and the modified cell was able to divide and multiply. To prove that the cell’s genome was artificially manufactured, the Venter group inserted four markers into the DNA sequence. The markers included the names of the researchers; a paraphrased quotation from Richard Feynman, “What I cannot create, I do not understand” (words found on his blackboard after he died); and a message congratulating the decoder.

The first use of DNA in computing was an experiment performed by Len Adleman, whom we met earlier in the discussion of the RSA encryption scheme. Adleman invented a way of using single-stranded DNA – one side of a DNA ladder with its sequence of bases not paired with a partner DNA strand – to solve a puzzle called the *seven-city directed Hamiltonian path problem*, a variation of the traveling salesman problem we discussed in Chapter 5. To solve a seven-city directed Hamiltonian path problem, you must find the shortest route between seven cities, beginning at one designated city and ending at another, passing through each of the other five cities exactly once. In Adleman’s experiment, a single strand of DNA represented each city, with a corresponding unique

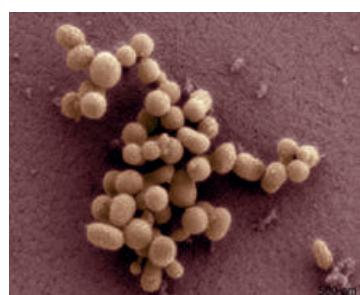
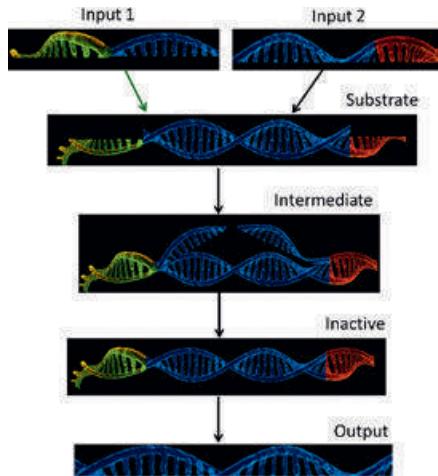


Fig. 15.23. The first self-replicating synthetic cells from the J. Craig Venter Institute.

Fig. 15.24. A DNA logic gate. This gate takes two DNA strands as input and only produces an output if both input strands are present.



sequence of A, T, C, and G. All the possible paths were represented by complementary DNA sequences consisting of the last half of a strand corresponding to a departure city and the first half of another strand representing a possible arrival city. Adleman mixed the DNA strands together and all the possible paths were generated, created by the complementary A-T and C-G bonding between the strands. He then had to perform a manual analysis to separate out molecules representing paths that did not start or end with the right city or paths that did not go through all the different cities. This DNA-based computation produced all possible paths very quickly due to the large number of molecules involved, but the manual analysis to separate out the strands for valid paths took several days. So although Adleman's work was an interesting experimental approach to computing with DNA, it was not a practical method for solving large-scale problems in a reasonable time.

An alternative direction for DNA computing has been to focus on creating more general-purpose computational circuits using both single-stranded and double-stranded DNA. The technique is called *strand displacement*, and the interactions are specified by the choice of complementary DNA sequences. A strand displacement reaction is initiated when a short portion of an incoming single strand binds to a complementary exposed portion of a double-stranded complex. If the remaining sequence of the incoming strand matches the sequence of a neighboring strand in the complex, the incoming strand will displace the existing strand through strand displacement. In this way, researchers have been able to create logic gates from DNA (see Fig. 15.24). Several research groups are experimenting to scale up such strand displacement gates to perform complex computations. The ultimate goal of this research is to make programming DNA circuits as straightforward as programming computers.

Key concepts

- Nanotechnology
- Memristor

- 3D transistor
- Carbon nanotubes
- Quantum computer
- Quantum entanglement
- DNA computing
- Strand displacement



16 The third age of computing

Every 30 years there is a new wave of things that computers do. Around 1950 they began to model events in the world (*simulation*), and around 1980 to connect people (*communication*). Since 2010 they have begun to engage with the physical world in a non-trivial way (*embodiment*).

Butler Lampson¹

The next revolution

The first age of computing was concerned with using computers for *simulation*. As we have seen, the first computers were built to do complex calculations. The initial motivation for building the ENIAC was to calculate artillery tables showing the angles at which guns should be fired based on the distance to the target and other conditions. After World War II, scientists used the ENIAC to explore possible designs for a hydrogen bomb. More generally, computers were used to simulate complex systems defined in terms of a mathematical model that captured the essential characteristics of the system under study. During the first thirty years of computing, from about 1950 until the early 1980s, researchers increasingly used computers for simulations of all sorts of complex systems. Computer simulations have transformed our lives, from designing cars and planes to making weather forecasts and financial models. At the same time, businesses used computers for performing the many, relatively simple calculations needed to manage inventories, payroll systems, and bank transactions. Even these very early computers could perform numerical calculations much, much faster than humans.

The second age of computing was about using computers for *communication*. The last thirty years, from the early 1980s until today, have seen computers become personal, not only for scientists and businesses but also for consumers. We now routinely use laptops, mobile phones, and tablets for a variety of activities, such as word processing, sending emails, searching the web, sharing photos, reading ebooks, and watching videos. Huge improvements in processing power together with astonishing miniaturization have come from the steady advance of computer technology predicted by Moore's law. It is these dramatic improvements in power and size during the last thirty years that have made possible the wide range of compact, portable computing devices we have available today. But this miniaturization of computing has been accompanied by an equally dramatic increase in *connectivity*, the ability to communicate with other computers.

Today's global Internet emerged, along with the increasing availability of wireless networks, from early experiments with the ARPANET, a computer network created by the U.S. Department of Defense in the late 1960s and the 1970s as a means of communication between research laboratories and universities. And by the 1990s, the World Wide Web had arrived to transform our online lives. The web made it possible for people with little computer proficiency to surf the Internet. It also enabled electronic commerce sites such as Amazon to emerge as serious competitors to bricks-and-mortar businesses. The increasing connectivity of our computing and communication devices has also led to the rise of social networking sites like Facebook and Twitter and the emergence of *crowdsourcing*, the practice of gathering services, ideas, information, or money by soliciting contributions from a large group of people online. Crowdsourcing websites include Amazon's Mechanical Turk, a service that uses humans to perform tasks that people do better than computers, such as comparing colors or translating foreign languages. Another is Wikipedia, a free encyclopedia that permits anyone to write and edit almost all its articles. Still another is Galaxy Zoo, an astronomy project that invites people to help classify large numbers of galaxies.

Butler Lampson's third age of computing is about using computers for *embodiment* – that is, using computers to interact with people in new and intelligent ways:

The most exciting applications of computing in the next 30 years will engage with the physical world in a non-trivial way. Put another way, computers will become *embodied*.²

He asserts that the present state of computer applications, such as robotic surgery, remote-controlled drones, robotic vacuum cleaners, and cruise controls for cars, are still in their infancy. In the next few decades, Lampson predicts, medical science will develop prosthetic eyes and ears that will enable people to really see and hear; cars will drive themselves; sensors in our homes and bodies will continuously monitor our health and well-being; and we will have intelligent, robot personal assistants to help us both at work and at home.

For computer systems to achieve such engagement, Lampson believes, they will have to handle uncertainty and probability as well as they now handle facts:

Probability is also essential, since the machine's model of the physical world is necessarily uncertain. We are just beginning to learn how to write programs that can handle uncertainty. They use the techniques of statistics, Bayesian inference and machine learning to combine models of the connections among random variables, both observable and hidden, with observed data to learn parameters of the models and then to infer hidden variables such as the location of vehicles on a road from observations such as the image data from a camera.³

In addition to managing uncertainty, many of the applications of embodiment will need to be much more dependable than today's computer systems. Computers driving cars or performing surgical procedures are obvious examples of applications in which reliability is critical for safety. We need methods



Fig. 16.1. George Devol's original patent for the first programmable robotic arm in 1954 was the foundation for the modern robotics industry.



B.16.1. George Devol (1912–2011) was the inventor of robotic arm. This programmable device became very successful and revolutionized the manufacturing industry.

for specifying the desired behavior of a computer program and proving that the resulting code actually fulfills these specifications. Today, such methods work only for small-scale systems, and there is still much research needed to scale these methods up to handle the large, critical safety applications of tomorrow.

In this chapter we will look at two key trends – the coming robotics revolution and the “Internet of Things” – and end the chapter with some words about consciousness and realistic neural networks.

The rise of the robots

The word *robot* was introduced to the world by the Czech author Karel Čapek in his play R.U.R., an abbreviation for Rossum’s Universal Robots. The play, first performed in 1921, portrayed mass-produced artificial humans that could manufacture products much more cheaply than real people could. The word *robot* is derived from the Czech word *robota*, meaning *labor*. The theme of R.U.R. is now a familiar one in science fiction, a revolt by the robots. Another science fiction writer, Isaac Asimov, introduced the word *robotics* to mean the science and technology of robots. Robotics is now an established field of research and requires a combination of many different disciplines, ranging from mechanical engineering and power systems to computer vision and machine learning.

The first industrial robot was a far cry from the humanoid robots imagined by Čapek and Asimov. In 1954, George Devol (B.16.1) invented a static, immobile machine with a programmable arm. His patent on a device for “programmed article transfer” issued in 1961 laid the foundation for the modern robotics industry (Fig. 16.1). In his patent application he wrote, “The present invention makes available for the first time a more or less general purpose machine that has universal application to a vast diversity of applications where cyclic digital control is desired.”⁴

Devol coined the phrase *universal automation* to describe a robot that could be programmed to perform a variety of tasks, and he called his first product the Unimate. The first Unimate machine was sold to General Motors in 1960. General Motors installed it at the auto-body plant in Ewing Township, New Jersey, to lift and stack hot pieces of metal from a die-casting machine. Devol’s next product was a robotic arm for spot welding. The early industrial robots did not look anything like the humanoid robots of science fiction; many consisted of little more than a mechanical arm. Other automobile companies soon followed General Motors’s lead, using robots to do jobs that were tedious, hard, or dangerous for people.

Around twenty thousand robots are now sold each year in North America. Although the U.S. automotive industry is still the dominant sector, sales also grew in the life sciences and pharmaceutical industry. Japan leads the world with an installed base of several hundred thousand industrial robots. These modern industrial robots are far more sophisticated than their early ancestors. On the new Tesla electric car assembly line in California, for example, at each station there are up to eight robots, some more than eight feet tall, each with a single arm with multiple joints, and each capable of multiple functions, such as welding, riveting, and bonding different components (see Fig. 16.2). Since

Fig. 16.2. Robots handling the delicate operation of glass panel unloading.



Fig. 16.3. Sony's doglike AIBO robots playing soccer at the 2005 RoboCup competition.



Fig. 16.4. Honda's ASIMO robot has appeared at conferences and toured the world since 2000.

the introduction of the early industrial robots, the number and type of robots have increased dramatically. We will look at four examples: humanoid robots, robotic laboratories, driverless vehicles, and drones.

Japanese companies have pioneered the development of animal-like and humanoid robots. In the 1990s, after a suggestion by a Canadian researcher, Alan Mackworth, Japanese researchers in artificial intelligence (AI) started an annual soccer competition for robots called the Robot World Cup, or RoboCup (Fig. 16.3). The aim of the RoboCup was to promote robotics and AI research. Playing soccer requires the robots not only to move and act independently but also to collaborate and follow a team strategy to beat the opposing team. As the robots play, they have to process many different types of sensor input and make real-time decisions based on this input. In 1999, Sony Corporation produced the AIBO – Artificial Intelligence roBOt – a four-legged, doglike robot designed to serve as a household pet. Teams of AIBOs have regularly competed in the RoboCup.

In 2000, the Honda Motor Company produced a humanoid robot called ASIMO (Advanced Step in Innovative Mobility), an acronym chosen to give homage to Isaac Asimov (Fig. 16.4). The robot is about four feet high and can detect movements of objects and recognize distance and direction using two camera “eyes.” ASIMO can also understand some voice commands and gestures, such as when a person offers to shake hands. In 2006, at the International Consumer Electronics Show in Las Vegas, Nevada, ASIMO demonstrated its ability to walk, run, and kick a football. Such experiments are not just research stunts. Because Japan has an aging population, robots of all sorts may serve as one possible way of assisting the elderly.

The National Aeronautics and Space Association (NASA) uses robotic geologists for its exploration of the surface of Mars. The Mars Exploration Rovers – Spirit and Opportunity – landed on Mars in 2004 and examined rocks and soils to find out the role that water has played in the history of Mars. These robots could drive up to forty meters a day and carried a range of scientific instruments, including a panoramic camera, various types of spectrometers, magnets, a microscope, and an abrasion tool for scratching rock surfaces. Curiosity, a much more ambitious mobile robotic laboratory, successfully landed on Mars in August 2012 (see Fig. 16.5). The Curiosity rover is about three meters long and five times as heavy as the previous rovers. Unlike the earlier vehicles, Curiosity can gather samples of rocks and soil and distribute them to onboard analytical instruments. Its mission is to investigate whether conditions on Mars have ever supported microbial life.



Fig. 16.5. NASA rovers began the robotic exploration of Mars in 2004.



Fig. 16.6. Carnegie Mellon University's driverless vehicle Sandstorm competed in the 2004 and 2005 DARPA Grand Challenges.

The first Defense Advanced Research Projects Agency (DARPA) Grand Challenge, held in the United States in 2004, was a competition for driverless vehicles funded by DARPA. The goal was to successfully navigate a 150-mile course in the Mojave Desert in California. No team successfully completed the course in the first Grand Challenge. The vehicle that traveled farthest was Sandstorm, a converted Humvee built by a research team from Carnegie Mellon University. Sandstorm covered more than seven miles before it caught fire and ended up stranded on a rock. No prize was given that year, and the organizers scheduled a second event a year later. That time, five vehicles finished the course. The winner was Stanley, built by a team from Stanford University led by Sebastian Thrun. Stanley completed the course in about seven hours, closely followed by two entries from Carnegie Mellon, Sandstorm and Highlander, led by the roboticist Red Whittaker (Fig. 16.6). In 2007, DARPA organized a third driverless car competition, this time on a sixty-mile course called the Urban Challenge that required driving through inhabited areas. The robotic vehicles had to avoid other vehicles and obstacles in a crowded urban environment, obeying all traffic regulations. The challenge was won by Tartan Racing, a team from Carnegie Mellon University, driving a modified Chevy Tahoe SUV named Boss.

The robotic vehicles most in the news are undoubtedly *unmanned aerial vehicles*, also called *drones* (Fig. 16.7). The military increasingly uses drones for surveillance and battlefield exploration. These military drones are typically large and expensive. Just as we saw the PC movement emerge from the hobbyist community, today we are seeing explosive growth of a low-cost "hobbyist" drone movement. A key ingredient for a drone is an autopilot. When autopilots were originally introduced in the 1930s, the control systems merely kept the aircraft level and flying on a preset course. Nowadays, autopilots can be used to automate the whole flight plan, as well as the takeoff and landing. What has changed in the last ten years is that all the components needed to construct an autopilot have become much smaller and cheaper. These devices include gyroscopes to measure rates of rotation; magnetometers to act as a digital compass; barometric pressure sensors to determine altitude; and



Fig. 16.7. The Northrop Grumman Global Hawk drone can fly at sixty thousand feet for flights as long as thirty hours.



Fig. 16.8. The MeCam quadcopter is a miniature drone that can follow and photograph you wherever you go and then upload the video images to your smart phone.

accelerometers to measure changes in motion. Chips with all these functionalities now cost less than \$20. Similarly, the demand for smaller global positioning system chips to provide navigation systems in phones has brought the price down from thousands of dollars to as little as \$10. Finally, the demand for better mobile phone cameras has led to the availability of cheap, powerful imaging chips. As a result, there is now a thriving do-it-yourself drone community. Hobbyist drone builders employ smart phone technology, including low-cost sensors, cameras, low-power processors, and batteries. Drone enthusiasts exchange information on DIYdrones.com, a website set up by Chris Anderson of Wired magazine (Fig. 16.8). The site lists a large number of nonmilitary, nonpolice uses of drones – including agriculture, search and rescue, home movies, coverage of sports events, environmental monitoring, and delivering medicines.

The Internet of Things

The growth of the Internet over the last thirty years has been dramatic. From connecting a few thousand computers at research centers, the Internet now connects billions of people through their personal computers, smart phones, and tablets. Yet this is only the first step in connectivity. We can now attach cheap electronic tags and sensors to objects and connect them to create an even larger global network called the “Internet of Things.” The MIT engineer Kevin Ashton first used the term in 1999. In his original definition, he said:

Today computers – and, therefore, the Internet – are almost wholly dependent on human beings for information. Nearly all of the roughly 50 petabytes (a petabyte is 1,024 terabytes [trillion bytes]) of data available on the Internet were first captured and created by human beings – by typing, pressing a record button, taking a digital picture or scanning a bar code.... The problem is, people have limited time, attention and accuracy – all of which means they are not very good at capturing data about things in the real world.... If we had computers that knew everything there was to know about things – using data they gathered without any help from us – we would be able to track and count everything, and greatly reduce waste, loss and cost. We would know when things needed replacing, repairing or recalling, and whether they were fresh or past their best. The Internet of Things has the potential to change the world, just as the Internet did. Maybe even more so.⁵

After the revolutions caused by the World Wide Web and mobile, networked, wireless devices, the Internet of Things represents the next disruptive technology on the horizon. With more than fifty billion objects predicted to be connected to the Internet by 2020, we will see a world in which everyday objects, such as books, cars, and refrigerators, can be interrogated for information. Some of the smart devices will not only use sensors to get information but also use *actuators*, devices that move or control things, to modify the environment. “Intelligent houses” will be able to check on their inhabitants; water drainage systems will know about storm threats and adjust accordingly; and businesses will actively monitor supply chains so that they no longer run out of stock or generate wasteful surpluses.

How will we be able to cope with the immense complexity of this new world? The computer entrepreneur Ray Ozzie has painted a vision of what he calls “a world of continuous services and connected devices”:⁶

To cope with the inherent complexity of a world of devices, a world of websites, and a world of apps and personal data that is spread across myriad devices and websites, a simple conceptual model is taking shape that brings it all together. We’re moving toward a world of 1) cloud-based *continuous services* that connect us and do our bidding, and 2) appliance-like *connected devices* enabling us to interact with those cloud-based services.⁷

Cloud computing is a way of sharing computing resources by linking large numbers of computers and other devices over the Internet with massive data centers where huge amounts of information can be stored together with massive, on-demand, computational capacity. Websites will use these cloud resources to provide continuous services that are always available and can be scaled to meet any fluctuation in demand. These services will constantly gather and analyze data from both the real and online worlds. Users will interact with these services using “apps” – software applications – on a range of connected devices. Increasingly, as the Internet of Things grows, these devices will include many types of embedded systems, from webcams in our homes to sensors on our highways.

Strong AI and the mind-body problem

WARNING: In the remaining sections of this chapter, we enter into areas in which there is no clear consensus among researchers, computer scientists, neuroscientists, and philosophers. There are many different opinions and often little agreement even about definitions.

In their book *Artificial Intelligence*, Stuart Russell and Peter Norvig define the terms *weak AI* and *strong AI* as follows:

The assertion that machines could act *as if* they were intelligent is called the *weak AI* hypothesis by philosophers, and the assertion that machines that do so are actually thinking (not just simulating thinking) is called the *strong AI* hypothesis.⁸

The proposal for John McCarthy’s 1956 workshop that introduced the term *AI* confidently asserted that weak AI was possible, saying, “Every aspect of learning or any other feature of intelligence can be so precisely described that a machine can be made to simulate it.”⁹

In their book, Russell and Norvig take the view that “intelligence is concerned mainly with rational action.”¹⁰ They introduce the idea of building intelligent systems in terms of *agents*, subsystems that can perceive their environment through sensors and can act on the environment through actuators. A rational agent is one that selects an action that maximizes its performance for every possible sequence of inputs. The agent can also learn from experience to improve its performance. Russell and Norvig identify different types of agents, including reflex agents, goal-based agents, and utility-based agents. *Reflex agents* respond only to their last input. *Goal-based agents* act to achieve a well-defined

goal, and *utility-based agents* try to maximize some specific measure of performance. Such rational agent-based systems have had considerable success during the last twenty years in robotics, speech recognition, planning and scheduling, game playing, spam fighting, and machine translation, to list only a few examples. Because of such progress, Russell and Norvig declare:

Most AI researchers take the weak AI hypothesis for granted, and don't care about the strong AI hypothesis – as long as their program works, they don't care whether you call it a simulation of intelligence or real intelligence.¹¹

In spite of this very pragmatic approach from the majority of AI practitioners, intelligent machines have continued to be an active topic of discussion by philosophers since Alan Turing devised his Universal Turing Machine in 1950. Francis Crick, one of the discoverers of DNA, has said that the scientific study of the brain during the twentieth century has led to the acceptance of consciousness as a valid subject for scientific investigation. In his 1994 book *The Astonishing Hypothesis*, Crick suggests that “a person's mental activities are entirely due to the behavior of nerve cells, glial cells, and the atoms, ions and molecules that make up and influence them.”¹² In other words, the human mind arises entirely from the actions of billions of neurons in the brain.

Ever since the days of Plato and Aristotle, philosophers have been concerned with the *mind-body* problem, which examines the relationship between mind and matter. René Descartes, in the seventeenth century, viewed the activity of thinking and the physical processes of the body as distinct – a philosophy known as *dualism*. By contrast, *monism* maintains that the mind and brain are not separate and that mental states are just physical states – a viewpoint sometimes described as *physicalism*.

Many philosophers and computer scientists are attracted to the idea of *functionalism*, in which a mental state is defined solely by its function – that is, its relation to sensory inputs, other mental states, and behavior. There are many varieties of functionalism, but we shall focus on Hilary Putnam's idea of *machine functionalism*, which makes an analogy between the states of a Turing machine and the mental states of the brain. As we have seen, the output of a Turing machine is determined by the initial state of the machine and the tape input. This is the basis of *computationalism*, the theory that mental states are just computational states and the transition from one mental state to another depends only on its inputs and is independent of the particular physical implementation. This viewpoint leads naturally to the question “Can machines think?” and to questions about strong AI.

The dominant trend in psychology in the first half of the twentieth century was an approach called *behaviorism*, championed by John Watson and B. F. Skinner. This movement maintained that psychology should be concerned only with observable behavior of people and animals and not with untestable, unobservable events that may or may not be taking place in their minds. Alan Turing's famous Turing Test, which we discussed in [Chapter 13](#), is a behavioral test for intelligence. In a response to this type of intelligence test, in 1980 philosopher John Searle introduced his famous “Chinese room” experiment to show that behavior is not enough for understanding and strong AI. We introduced Searle's Chinese room experiment in [Chapter 14](#) in the context of IBM's

Watson's victory on *Jeopardy!* In more detail, his thought experiment tests the thesis that human *cognition* – thinking, understanding, and feeling – is nothing more than computation. Searle's argument is very simple:

Because it is impossible to know whether anyone else but myself cognizes (thinks, understands, feels) I cannot say whether that computer over there, successfully passing the Turing Test [TT] in Chinese, is really cognizing (i.e. understanding Chinese). However, because computation is implementation-independent – which means that every implementation of the same computer program should have the same properties, if the properties are truly just computational ones – then I, Searle, can become an implementation of the Chinese-TT-passing program too. Yet it is evident, even without doing the experiment, that if I did so, I would not be understanding Chinese: I would just be manipulating meaningless symbols, according to the formal rules I had memorized – squiggles and squoggles.... So whereas I still cannot say whether the TT-passing computer understands Chinese, I can say for sure that if it is understanding Chinese, it is not because it is implementing the right computer program. This is because I am implementing the same computer program, and I am definitely not understanding Chinese. So computationalism (strong AI) is false (or incomplete).¹³

The cognitive scientist Stevan Harnad (B.16.2) argues further that there is something else missing in the Chinese room, besides the question of Searle's understanding of Chinese:

Not only does he not understand the meaning of the symbols he is manipulating, but he also cannot pick out their referents. If you ask him what “BanMa” (“zebra” in Chinese) means, he will not only say, correctly, that he has no idea (even though he has just got done stating, *in Chinese*, that “A BanMa looks like a striped horse”). But apart from the missing feeling of understanding, Searle also cannot pick out the thing that BanMa refers to in the world: the symbols are ungrounded. Grounding (for which you need more than computation – you need robotic sensorimotor interactions with the world, to learn what symbol refers to what object) is necessary, though not sufficient, for meaning. In addition, it also feels like something to mean (or understand) BanMa.¹⁴



B.16.2. Stevan Harnad was born in Budapest, Hungary. He currently holds a Canadian Research Chair in cognitive science at the Université du Québec in Montreal and is also professor of cognitive science at the University of Southampton, England. He has championed the need for a *Total Turing Test* for which the standard Turing Test is extended to include the computer's perceptual and manipulative abilities.

Harnad's introduction of the *symbol grounding problem* is based on the following argument. Computation is the manipulation of symbols based on the symbols' shapes, not their meanings. Computation alone does not and cannot connect symbols to their meanings, as it would have to do for computation to be cognition. Harnad believes that some of the symbols have to be grounded in the sensory and motor capacity to pick out their corresponding objects in the world. He has therefore proposed extending the traditional Turing Test to the *Total Turing Test*, which includes a test of the computer's perceptual and manipulative abilities that are not purely computational. From this point of view, Searle's Chinese room merely shows that computation alone is insufficient for cognition.

The British mathematician Roger Penrose has put forward another objection to strong AI and computationalism. Penrose's argument is based on the logician Kurt Gödel's finding that there are “nonalgorithmic truths,” statements

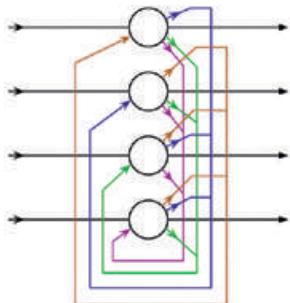


Fig. 16.9. A four-node Hopfield network with feedback loops.

that humans know to be true but that cannot be proved within a formal system based on a set of axioms. Penrose claims that this finding shows that computers, which can only operate by following algorithms, are therefore necessarily more limited than humans. This argument has been the subject of much debate by many people, including Turing. He observed that such results from mathematical logic could have implications for the Turing Test:

There are certain things that [any digital computer] cannot do. If it is rigged up to give answers to questions as in the imitation game, there will be some questions to which it will either give a wrong answer, or fail to give an answer at all however much time is allowed for a reply.¹⁵

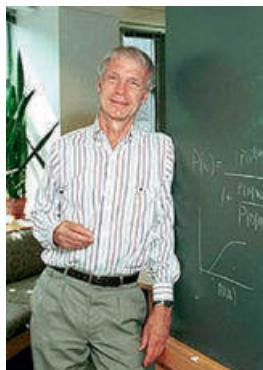
In the context of the Turing Test, the existence of such nonalgorithmic truths implies the existence of a class of “unanswerable” questions. However, Turing asserted that these questions are only a concern for the Turing Test if humans are able to answer the same questions.

Neural networks revisited

Rather than delve deeper into these hotly contested, largely philosophical issues, we shall look again at what the brain might tell us about intelligence and consciousness. We start with another look at neural networks. In the body, a neural network consists of interconnected nerve cells that work together, such as in the brain. In computer science, a neural network is a network of electronic components loosely modeled on the operation of the brain. As we have seen, the artificial neural networks (ANNs) described in Chapter 13 have successfully performed many pattern recognition tasks.

These ANNs, however, are very far from functioning like a realistic neural network in a living organism. Besides the huge difference in the numbers of neurons and connections, the primary element lacking is that of *feedback*, in which information is sent back into the system to adjust behavior. The favored method of training the ANN is *back propagation*, in which the initial output is compared to the desired output, and the system is adjusted until the difference between the two is minimized. However, the ANN was purely a *feed-forward* network that produced a specific output for each given set of inputs. In real brains, nerve cells not only feed forward but also send information back to other neurons.

An example of an ANN that allows feedback is the *Hopfield network* (Fig. 16.9), named after the multidisciplinary scientist John Hopfield (B.16.3). This network introduces bidirectional connections between the artificial neurons and assumes that the weights for each connection are the same in each direction. Such neural networks are able to function as *auto-associative memories* – that is, when a pattern of activity is presented to the network, the neurons and connections form a memory of this pattern. Even if you only input a part of the original pattern, the auto-associative memory can retrieve the entire original pattern. It is also possible to design these networks to store temporal sequences of patterns, capturing the order in time in which they occur. Feeding in only a part of this sequence generates the whole sequence, just as hearing the first few notes of a song brings back the whole song. The computer architect Jeff



B.16.3. John Hopfield was originally trained as a physicist but is most widely known for his research on ANNs. He was responsible for setting up the Computation and Neural Systems PhD program at Caltech and is now the Howard A. Prior Professor of Molecular Biology at Princeton.

Hawkins ([B.16.4](#)) has built on these ideas of neurons and memory and proposed an alternative model of the brain to the computationalist view discussed in the preceding text. We briefly outline some of the key ideas of his *memory-prediction theory* of intelligence.



[B.16.4.](#) Jeff Hawkins is a computer entrepreneur most known for his work on handheld computing devices such as the Palm Pilot and the Treo. He invented the handwriting character recognition system known as *Graffiti* for use with such devices. In addition to his successful career in the computer industry Hawkins has a deep interest in the function of the brain and wrote the book *On Intelligence* describing his *memory-prediction framework* of how the brain works.

Hawkins believes that any model of the brain and intelligence needs to incorporate neurons with feedback and be able to respond to rapidly changing streams of information. He focuses his attention on the architecture of the human cortex, the part of the brain responsible for higher functions, such as voluntary movement, learning, and memory. As we described in [Chapter 15](#), the cortex is about 2.5 millimeters deep and is made up of six layers, each about as thick as a playing card. The cortex is estimated to contain around thirty billion neurons, each with thousands of connections making a total of more than thirty trillion synapses, the junctions at which nerve impulses pass from one neuron to another. Neurologists have found that the cortex consists of many different functional regions, each semi-independent and specialized for certain aspects of thought and perception. Each region is arranged in a hierarchy, with “lower” areas feeding information up the hierarchy and “higher” areas sending feedback back down toward the lower layers, although the terms *higher* and *lower* are not necessarily related to their physical arrangement in the brain. The lowest areas are the primary sensory areas, where sensory information arrives. The cortex has regions to process sensations from the eyes, the ears, and the skin and internal organs, and each region has its own hierarchies of regions. The cortex also has “association” areas where inputs from more than one sense can be combined. There is also a motor system in the frontal lobes of the brain that sends signals to the spinal cord and thus moves muscles. The hierarchies of all these sensory areas look very similar. This similarity led Vernon Mountcastle ([B.16.5](#)), a neuroscientist from Johns Hopkins University in Baltimore, to propose a model for the basic structure of the cortex in a paper titled “An Organizing Principle for Cerebral Function.”

In 1950, Mountcastle had discovered that the cortex was organized into vertical columns of neurons, with each column having a particular function. In 1978, he proposed that all parts of the cortex operate on a common principle, with the cortical column being the fundamental computational unit ([Fig. 16.10](#)). All the inputs from our primary sensory areas arrive at the cortex as patterns of partly chemical and partly electrical signals. We rely on our brains to make sense of this stream of data and to produce a consistent and stable view of the world. For example, several times a second, our eyes make sudden movements called *saccades*. With these saccades, the focus of our eyes moves around, locating interesting parts of the scene so that our brain can build up a three-dimensional model of what we are seeing ([Fig. 16.11](#)). Our impression of a stable world with objects and people moving in a continuous way is only possible because our brain has the processing capability to make sense of this continuous stream of changing retinal patterns ([Fig. 16.12](#)). Mountcastle speculated that all neurons in the cortex use the same basic algorithm to process the different input patterns arriving at the different sensory input regions – those for vision, hearing, language, motor control, touch, and so on. In other words, the brain processes patterns and constructs a model of the world that it then holds in memory made up of neurons and their synapses.



[B.16.5.](#) Vernon Mountcastle is Professor Emeritus at Johns Hopkins University. He is best known for his discovery of the columnar organization of the cerebral cortex in the 1950s. In 1978 he proposed that all parts of the cortex operate on a common principle based on these cortical columns.

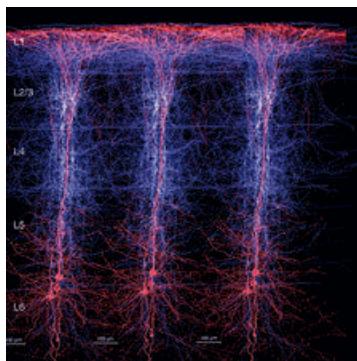


Fig. 16.10. Visualization of cortical columnar structure discovered by Vernon Mountcastle. Jeff Hawkins describes Mountcastle's 1978 paper proposing that all cortical columns operate on a common principle as the *rosetta stone* of neuroscience.



Fig. 16.11. The human brain uses the tracks of saccades, sudden movements made by the eyes when observing a face, to build up a three-dimensional model of what the eyes see.



Fig. 16.12. A one-day-old baby, appropriately called Ada, is building up her picture of the world, three images per second, every second of her waking life.

Up to now, we have considered Turing's behavioral test for intelligence as the basis for the computationalist model of the brain, which views the brain as a computer running programs. Hawkins makes two points in criticism of this viewpoint. First, there is what he calls the *input-output fallacy*: the behaviorist view that you present the brain with a given input and observe what output you get. In fact, when our brains receive input, we may indeed process that data and perform a visible action, but we do not necessarily respond that way. The input can just lead to thoughts that are not expressed in actions. Actions are optional, and this aspect of intelligence is not captured by a purely behavioral test. The second criticism concerns what Hawkins calls the *one hundred-step rule*. In the computer analogy of the brain, it is customary to contrast the one hundred billion neurons in a human brain with the few billion transistors on a chip. By contrast, a typical neuron takes about 5×10^{-3} seconds (5 milliseconds) to fire and reset compared to the cycle time for a modern chip which can be as short as 5×10^{-9} seconds, about a million times faster than a neuron. To account for the amazing power of our brain, given the relative slowness of the individual neurons, computationalists point to the fact that billions of neurons can be computing at the same time, as in a parallel computer that uses more than one CPU to execute a program, making it run faster. But consider the problem of looking at a photograph to determine whether there is a cat in the image. A human can pick out any cat in the photograph in less than a second. However, in that second, because neurons operate so slowly, the visual information entering the brain can only cross a chain of about a hundred neurons or so from the visual sensory input region. Thus the brain must "compute" its answer using only a tiny fraction of its billions of neurons. By contrast, to solve such a cat recognition problem on a digital computer would take many billions of steps.

How can a brain perform a difficult recognition task in only a hundred steps that would take a supercomputer many billions of steps? Hawkins suggests:

The answer is the brain doesn't "compute" the answers to problems; it retrieves the answers from memory. In essence, the answers were stored in memory a long time ago. It takes only a few steps to retrieve something from memory. Slow neurons are not only fast enough to do this, but they constitute the memory themselves. The entire cortex is a memory system. It isn't a computer at all.¹⁶

Let us give one last example to show how the brain handles the task of catching a ball. Someone throws a ball toward you and less than a second later you catch it. If we want to program a robot arm to catch the same ball, the program requires an enormous amount of computation. First, you have to estimate the trajectory of the ball and calculate it numerically by solving Newton's laws of motion. This calculation tells you roughly where to position the robot arm to catch the ball. Because the first calculation of the ball's trajectory was only an estimate, the whole calculation needs to be repeated several times as the ball gets nearer. Finally, the fingers of the robotic arm need to be programmed to actually close around the ball when it arrives. To accomplish all this, a computer requires many millions of steps to solve the numerous mathematical equations involved. Yet our brain uses its neurons to catch the

ball in only about a hundred steps. The brain clearly solves the problem a different way than relying on conventional computation. According to Hawkins, it uses memory:

How do you catch the ball using memory? Your brain has a stored memory of the muscle commands required to catch a ball (along with many other learned behaviors). When a ball is thrown, three things happen. First, the appropriate memory is automatically recalled by the sight of the ball. Second, the memory actually recalls a temporal sequence of muscle commands. And third, the retrieved memory is adjusted as it is recalled to accommodate the particulars of the moment, such as the ball's actual path and the position of your body. The memory of how to catch a ball was not programmed into your brain; it was learned over years of repetitive practice, and it is stored, not calculated in your neurons.¹⁷

To account for the fact that the position of the ball needs to be constantly adjusted as the ball comes toward us, Hawkins uses the idea that the memories stored in the cortex are actually *invariant representations*. Artificial auto-associative memories can recall complete patterns when given only a partial image as input. But ANNs have a hard time recognizing a pattern if the pattern has been rescaled, rotated, or viewed from a different angle – a task our brains can handle with ease. If you are reading a book, you can change your position, rotate the book, or adjust the lighting, so that the visual input of the book to your brain can be constantly changing. Yet your brain knows that the book is the same, and its internal representation of “this book” does not change. The brain’s internal representation is therefore called an *invariant representation*. The brain combines such invariant representations with changing data to make predictions of how to perform tasks, such as catching a ball.

Our understanding of the world is tied to our ability to make such predictions. Our brain receives a constant stream of patterns from the outside world, stores them as memories, and makes predictions by combining what it has seen before with the incoming stream of information. Hawkins says:

Thus intelligence and understanding started as a memory system that fed predictions into the sensory stream. These predictions are the essence of understanding. To know something means that you can make predictions about it.¹⁸

This idea is the basis of Hawkins’s *memory-prediction framework* of intelligence: “Prediction not behavior is proof of intelligence.”¹⁹ According to this view of intelligence, intelligent machines could be built that have just the equivalent of a cortex and a set of input sensors. There is no need to connect to the emotional systems of the other, older regions of the brain. Such intelligent systems will not resemble the humanoid robots of science fiction but would be able to develop an understanding of their world and make intelligent predictions. However, the technical challenges of building such systems in silicon still remain formidable, both in terms of the number of neurons required and their vast connectivity requirements.



B.16.6. Daniel Dennett is a philosopher and cognitive scientist who has written popular books on evolution and consciousness – *Darwin's Dangerous Idea* and *Consciousness Explained*. He is codirector of the Center for Cognitive Studies at Tufts University in Massachusetts.



B.16.7. Christof Koch was a professor of neuroscience at Caltech and, since 2011, the Chief Scientific Officer at the Allen Institute for Brain Science. During the 1990s, Koch collaborated with the late Nobel Prize recipient Francis Crick on the problem of consciousness as a scientifically addressable problem. He and Crick co-authored the 2004 book *The Quest for Consciousness: A Neurobiological Approach*.

Consciousness?

In their discussions of consciousness, philosophers often introduce the idea of a “zombie” as a category of imaginary human being. The philosopher Daniel Dennett (B.16.6) says:

According to common agreement among philosophers, a zombie would be a human being who exhibits perfectly natural, alert, loquacious, vivacious behavior but is in fact not conscious at all, but rather some sort of automaton. The whole point of the philosopher’s notion of a zombie is that you can’t tell a zombie from a normal person by examining external behaviors.²⁰

Philosophers also frequently introduce the idea of *qualia*, the plural of *quale*, into their discussions of consciousness. Neuroscientist Christof Koch (B.16.7) explains the concept of qualia as follows:

What it feels like to have a particular experience is the quale of that experience: The quale of the color red is what is common to such disparate percepts as seeing a red sunset, the red flag of China, arterial blood, a ruby gemstone, and Homer’s wine-dark sea. The common denominator of all these subjects is “redness.” Qualia are the raw feelings, the elements that make up any one conscious experience.²¹

In his attempt to move the debate about consciousness from a philosophical level to a legitimate topic for scientific investigation, Koch introduces four different definitions of consciousness:

A commonsense definition equates consciousness with our inner, mental life ...

A behavioral definition of consciousness constitutes a checklist of actions or behaviors that would certify as conscious any organism that could do one or more of them ...

A neuronal definition of consciousness specifies the minimal physiologic mechanisms required for any one conscious sensation ...

A philosopher [will] give you a fourth definition, “consciousness is what it is like to feel something.”²²

However, Dennett, in his book *Consciousness Explained*, takes a different approach and explicitly abandons the arguments and debates about qualia and avoids this concept in his own discussion of consciousness.

It is the ability to be self-aware that probably embodies what most people think is the essence of consciousness. Nevertheless, as we have seen, there is still a long way to go before computer scientists, cognitive scientists, and neuroscientists are ready to reach a consensus about strong AI, the mind-body problem, and consciousness. As philosopher Dennett has said, “Human consciousness is just about the last surviving mystery.”²³

Key concepts

- Humanoid robots
- Unmanned aerial vehicles

- Cloud computing
- The mind-body problem
- Strong AI
- Agents
- Functionalism
- Computationalism
- Behaviorism
- Symbol grounding problem
- Back propagation
- Auto-associative memories
- One hundred-step rule
- Consciousness



17 Computers and science fiction – an essay

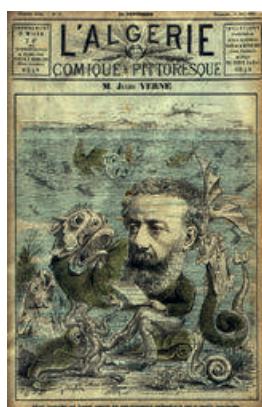
No one saw these mice coming. No one, that is, in my field, writing science fictions. Oh, a few novels were written about those Big Brains, a few New Yorker cartoons were drawn showing those immense electric craniums that needed whole warehouses to THINK in. But no one in all of future writing foresaw those big brutes dieted down to fingernail earplug size so you could shove Moby Dick in one ear and pull Job and Ecclesiastes out the other.

Ray Bradbury¹

Early visions

The British science fiction writer, Brian Aldiss, traces the origin of science fiction to Mary Shelley's *Frankenstein* in 1818. In her book, the unwise scientist, Victor Frankenstein, deliberately makes use of his knowledge of anatomy, chemistry, electricity, and physiology to create a living creature. An alternative starting point dates back to the second half of the nineteenth century with the writing of Jules Verne (B.17.1) and Herbert George (H. G.) Wells (B.17.2). This was a very exciting time for science – in 1859 Charles Darwin had published the *Origin of Species*; in 1864 James Clerk Maxwell had unified the theories of electricity and magnetism; and in 1869 Mendeleev had brought some order to chemistry with his *Periodic Table of the Elements*, and Joule and Kelvin were laying the foundations of thermodynamics. Verne had the idea of combining modern science with an adventure story to create a new type of fiction. After publishing his first such story "Five Weeks in a Balloon" in 1863, he wrote:

I have just finished a novel in a new form, a new form – do you understand? If it succeeds, it will be a gold mine.²



B.17.1. Jules Verne (1828–1905) was one of the founding fathers of the science fiction genre. This is the cover page of *L'Algérie* magazine from 15 June 1884, announcing his latest novel.

And a gold mine it turned out to be. In the next decade Verne published his highly successful series of *Voyages Extraordinaire* – *Journey to the Center of the Earth*, *From the Earth to the Moon*, *Twenty Thousand Leagues under the Sea*, and *Around the World in Eighty Days*. Verne's stories were grounded in scientific fact and set in the present or near future. Wells's approach to his *scientific romances* was in contrast to that of Verne. In 1895, Wells's novel, *The Time Machine*, has the scientist-inventor traveling through time, exploring a rather bleak future for humanity.

Two years later, in 1897, *The Invisible Man* has the researcher discovering a technique to make himself invisible – but is unable to reverse the process, leading to dire consequences. And in 1898, *The War of the Worlds* has alien invaders from Mars landing in London and defeating all of England’s military might with deadly heat-ray weapons. The runaway success of these stories, produced in just three years, has led to Wells being acclaimed as the “Father of Science Fiction.”

The nineteenth century of Verne and Wells was generally a time of optimism and of belief in the self-evident benefits of advances in science and technology. One early exception to this utopian view of the future was a short story by the novelist E. M. Forster. In 1909, he wrote *The Machine Stops*, which put forward a much more pessimistic, dystopian vision. In the story, humanity has become totally dependent on a vast and complex machine – which we would now call a computer – that manages every aspect of a completely mechanized environment for society. The story centers on a woman, Vashti, and her son Kuno. Vashti is content to live her life in her underground, automated apartment, attending meetings and lectures by videoconference, and only rarely physically meeting people:

Then she generated the light, and the sight of her room, flooded with radiance and studded with electric buttons, revived her. There were buttons and switches everywhere – buttons to call for food, for music, for clothing. There was the hot-bath button, by pressure of which a basin of (imitation) marble rose out of the floor, filled to the brim with a warm deodorized liquid. There was the cold-bath button. There was the button that produced literature, and there were of course the buttons by which she communicated with her friends. The room, though it contained nothing, was in touch with all that she cared for in the world.³

Kuno rebels against the uniformity of the society and tries unsuccessfully to go outside, into the open air. Total dependence on the Machine had almost become a religion. Unfortunately, the number of engineers who understood the technical workings of the Machine in its totality was decreasing every year. Eventually, increasing numbers of small defects became common – the music feeds became unreliable, the bath began to smell – and these were not immediately fixed by the Committee of Mending Apparatus. Finally, the Machine



B.17.2. Herbert George Wells (1866–1946), better known as H. G. Wells, was inspired by Darwin’s theory of evolution by natural selection as a student. The theme of humanity evolving according to these inexorable forces is evident in his first and probably most famous book, *The Time Machine*, published in 1895, as well as in many of his later works. He is often credited with foreseeing the development of the tank, aircraft and air warfare, the atomic bomb, and nuclear stalemate. Hugo Gernsback reprinted almost all of Wells’s novels in his *Amazing Stories* magazine and as a result Wells’s work was enormously influential on the development of science fiction in the United States. His book *The War of the Worlds* was the first alien invasion story, and Orson Welles’s radio dramatization of it caused a riot in New York in 1938. Toward the end of his life he had the idea of a distributed encyclopedia that he called the World Brain.



Fig. 17.1. The robot Wall-E was left behind to clean up the Earth's garbage after the humans had departed in spaceships. The movie *Wall-E* portrayed a world in which humanity had become entirely dependent on technology for their every need.

stops and with it, the dependent civilization. The movie *Wall-E* (Fig. 17.1) is a modern incarnation of this vision of humanity becoming overdependent on machines.

In his later life, Wells wrote several nonfiction books, including two major historical surveys – *Outline History* in 1920, and *The Work, Wealth and Happiness of Mankind* in 1932. In the course of writing these books, Wells realized the need for authors to have “information at their fingertips.” He believed that combating ignorance by making information readily available to the masses would reduce the likelihood of war. In 1937 he campaigned for the creation of a “World Brain” that would contain, and continually update, the knowledge contained in all the world’s great libraries, museums, and universities:

A World Encyclopaedia no longer represents itself to a modern imagination as a row of volumes printed and published once for all, but as a sort of mental clearing house for the mind, a depot where knowledge and ideas are received, sorted, summarized, digested, clarified and compared. It would be in continual correspondence with every university, every research institution, every competent discussion, every survey, every statistical bureau in the world. It would develop a directorate and a staff of men of its own type, specialized editors and summarists. They would be very important and distinguished men in the new world. The Encyclopaedic organization need not be concentrated now in one place; it might have the form of a network. It would centralize mentally but not physically.... It would constitute the material beginning of a real World Brain.⁴

Vannevar Bush and J. C. R. Licklider may have laid the foundations for today’s World Wide Web and the Internet, but Wells was one of the first evangelists for these ideas. With the Internet, the web, and Wikipedia, Wells would have been delighted by the progress that has been made toward his vision of a World Brain. Sadly, Wells was never able to realize the World Brain vision in his lifetime. After the terrible carnage of World War II, Wells descended into pessimism: his last book, *Mind at the End of Its Tether*, was written in 1945.

Wells never wrote about computers but Jules Verne did in *Paris in the Twentieth Century*, a novel published in 1994, a hundred and thirty years after it was written. The Paris of Verne’s novel had some remarkable insights – great avenues filled with horseless carriages, elevated railways and driverless trains, electric lighting in the shops and streets, and mechanical elevators in all the great buildings. His editor rejected the manuscript with the words “No one today will believe your prophecy”!⁵ Michel, the hero of the novel, is an apprentice at one of the huge banks that dominated the financial landscape:

Michel turned around and discovered the calculating machine behind him. It had been several centuries since Pascal had constructed a device of this kind, whose conception had seemed so remarkable at the time.... The Casmodge Bank possessed veritable masterpieces of the genre, instruments which indeed did resemble huge pianos: by operating a sort of keyboard, sums were instantaneously produced, remainders, products, quotients, rules of proportion, calculations of amortization and of interest compounded for infinite periods and all possible rates.⁶



Fig. 17.2. The cover of the April 1926 issue of *Amazing Stories*. It contained reprints of stories by H. G. Wells, Jules Verne, and Edgar Allan Poe.

The bank was also connected by telegraph to other institutions according to the Wheatstone telegraph system long since in use throughout England, and stock prices were continually updated from stock markets all around the world.

Further, photographic telegraphy, invented in the last century by Professor Giovanni Caselli of Florence, permitted transmission of the facsimile of any form of writing or illustration, whether manuscript or print, and letters of credit or contracts could now be signed at a distance of five thousand leagues.⁷

The Wheatstone telegraph system was in widespread use in Britain by the 1850s but Professor Giovanni Caselli had only just invented his facsimile *pantelgraph* system for transmitting images over telegraph lines. It was in 1862, one year before Verne wrote his book, that the first *pantelegram* had been sent from Lyon to Paris.

Computers and hard SF – the early years

It was the scientific romances of Wells and Verne that provided the inspiration for the first popular magazine in the United States dedicated to science fiction (SF). *Amazing Stories* was founded in 1926 by Hugo Gernsback (B.17.3). The first issue set the direction for the new magazine by republishing stories from Verne and Wells (Fig. 17.2). Gernsback was first to use the term *science fiction* to describe what he called “a charming romance intermingled with scientific fact and prophetic vision.”⁸ During the 1930s and 1940s, there was a great increase in the number of science fiction magazines but the most influential of all of these was undoubtedly *Astounding Science Fiction*. Its editor, John W. Campbell Jr. (B.17.4) had been a student at MIT and been taught by Norbert Wiener. Campbell’s own short story “The Last Evolution,” published in *Amazing Stories* in 1932, envisioned a future where man and machines fought together against an invasion from outer space. We can recognize aspects of modern computers in his description of the machines:

Machines – with their irrefutable logic, their cold precision of figures, their tireless, utterly exact observation, their absolute knowledge of mathematics – they could elaborate any idea, however simple its beginning, and reach the conclusion.⁹

As editor, Campbell set about raising the standard of writing in *Astounding*, training a whole generation of science fiction writers, including Asimov and Robert Heinlein. Although Campbell rejected Asimov’s first story, his rejection letter inspired Asimov to do better:

The joy of having spent an hour or more with John Campbell, the thrill of talking face to face and on even terms with an idol, had already filled me with the ambition to write another science fiction story, better than the first, so that I could try him again. The pleasant letter of rejection – two full pages – in which he discussed my story seriously and with no trace of patronization or contempt, reinforced my joy.¹⁰



B.17.3. Hugo Gernsback (1884–1967) was born in Luxembourg and emigrated to America when he was twenty. He published his novel *Ralph 124C 41+*, subtitled *A Romance of the Year 2660*, in his magazine *Modern Electrics* in 1911. In 1926, Gernsback launched *Amazing Stories*, the first magazine to be exclusively devoted to science fiction. The slogan on the title page proclaimed its mission “Extravagant Fiction Today ... Cold Fact Tomorrow.” Gernsback invented the term *science fiction* and the annual science fiction Hugo awards are named in his honor.



B.17.4. John Wood Campbell Jr. (1910–71) was indisputably the greatest editor of science fiction and nurtured a whole generation of science fiction writers including Isaac Asimov and Robert Heinlein. In 1938 he became editor of the magazine *Astounding Science Fiction* and, by his insistence on a much higher standard of writing and his help and support for writers like Asimov, created the modern science fiction genre. Under the pseudonym of Don A. Stuart he also wrote science fiction. Isaac Asimov rated Campbell's story "Who Goes There?," published in 1938, as "one of the very best science fiction stories ever written." (Photo courtesy of Marsh Library.)

Campbell was editor of *Astounding Science Fiction* and its successor, *Analog Science Fact – Science Fiction*, for more than thirty years. In a 1946 editorial, he gave a definition of what is now called *hard SF*:

Science fiction is written by technically minded people, about technically minded people, for the satisfaction of technically minded people.¹¹

In the early days of computers, after World War II, the public learned about the awesome calculating power of the ENIAC. Soon after this, Eckert and Mauchly launched the first U.S. commercial computer – UNIVAC. During the 1952 U.S. presidential election, UNIVAC shot to fame by appearing on TV and, with only a fraction of results reported, correctly predicted a landslide win for the preelection underdog, Dwight D. Eisenhower.

Given the enormous scale of these early vacuum tube computers it was natural for science fiction writers to equate the power of a computer with its size. Kurt Vonnegut (B.17.5) published a short story in 1950 about a computer called "EPICAC," the largest and smartest computer on the planet:

EPICAC covered about an acre on the fourth floor of the physics building at Wyandotte College. Ignoring his spiritual side for a minute, he was seven tons



B.17.5. Kurt Vonnegut (1922–2007) was a U.S. novelist who used science fiction techniques in many of his novels. His novel *Player Piano* explores the theme of computers making workers redundant, a scenario that is even more relevant today. This photograph of the Vonnegut mural in Indianapolis was created by the artist Pamela Bliss. Vonnegut's face in the mural is a composite image based on three different photos.

of electronic tubes, wires, and switches, housed in a bank of steel cabinets and plugged into a 110-volt A.C. line just like a toaster or a vacuum cleaner.¹²



B.17.6. Isaac Asimov (1920–92) was one of the most prolific science fiction authors of the twentieth century. His robot stories and his Three Laws of Robotics have defined the robot science fiction genre. A little-known fact about him is that he was also a professor of biochemistry.

The machine had been designed to do the same sort of tasks for the military as the ENIAC but in the story, EPICAC is sentient and falls in love with one of the mathematicians. In the end the machine blows itself up, leaving a suicide note and some love poems as a wedding gift for his human rival.

Isaac Asimov (B.17.6) also assumed that the most powerful computers of the future would be bigger than the ENIAC.

It wasn't until after computers were invented and the public was made aware of their existence, that computers began to exist in my stories, and even then I didn't truly conceive of the possibility of miniaturization.... In "The Last Question" I began with my usual computer, Multivac, as large as a city, for I could only conceive a larger computer by imagining more and more vacuum tubes heaved into it.¹³

In his 1955 short story “Franchise,” MULTIVAC is a government-run computer half a mile long and three stories high. The story is a satire on the use of UNIVAC to predict the result of the recent presidential election from a small sample of voters. In Asimov’s version, the sample size has been reduced to finding the views of the “Voter of the Year” – the single most representative person in the United States. In another MULTIVAC story, “The Machine That Won the War,” Asimov reveals some keen insight into the reliability of computers. The official propaganda is all about how the powerful MULTIVAC computer helped win the war for the Solar Federation. Yet when the three men whose job it was to interact with the computer met and discussed their roles, each admitted falsifying part of the computational process. The chief programmer admitted altering the data being fed to MULTIVAC because the people could not be trusted to supply accurate information in the chaos of war. The engineer then confessed to changing the data produced by the computer because he knew that MULTIVAC was not working reliably due to a shortage of manpower and spare parts. Finally, the Executive Director of the Solar Federation revealed that he had not trusted the reports produced by the machine and had made critical strategic war decisions by tossing a coin!

In 1961, the BBC in Britain broadcast a science fiction TV drama about a computer whose design originated from outer space. *A for Andromeda* was cowritten by TV producer John Elliot and the famous cosmologist Fred Hoyle (B.17.7). Using their newly commissioned radio telescope, researchers detect a signal coming from the Andromeda galaxy. The scientist hero, John Fleming, despite having to contend with constant interference from politicians and the military, is able to decipher the signal and deduce that it contains the instructions for building a new type of advanced computer. Fleming fears that the motivation for those sending the message is ultimately malevolent, but the military and the politicians insist that the computer be built and, as Fleming predicted, bad things happen.

One writer who did envision a future with smaller and more pervasive computers was Murray Leinster (B.17.8). His short story “A Logic Named Joe” appeared in the March 1946 issue of *Astounding Science Fiction*. The story is narrated by a



B.17.7. Fred Hoyle (1915–2001) was a celebrated cosmologist who worked at Caltech in the United States and Cambridge in the United Kingdom. He was one of the originators of the Steady State theory alternative to the Big Bang theory to explain the expansion of the universe – which has now been ruled out by experimental measurements. He also wrote several successful science fiction novels such as *The Black Cloud* and co-authored the BBC TV series *A for Andromeda*. The TV series featured Julie Christie in her first starring role.



B.17.8. Murray Leinster (1896–1975) was one of the few science fiction writers to envision the widespread availability of networked personal computers. His 1934 novel *Sideways in Time* is credited as being the first “parallel universe” science fiction story. In his 1945 novella *First Contact* Leinster introduced the idea of a universal translator, which is still some way from being a reality.



B.17.9. Gene Roddenberry (1921–91) is best known as the creator of the hugely successful *Star Trek* franchise. Before turning to screenwriting, Roddenberry had a wide variety of careers, from fighter pilot in the U.S. Air Force to a stint with the Los Angeles Police Department. In 1985 he became the first TV writer with a star on the Hollywood Walk of Fame.

maintenance man from the Logics Company and their “logic computers” had much the same capability as one of today’s networked personal computers:

You know the logics setup. You got a logic in your house. It looks like a vision receiver used to, only it’s got keys instead of dials and you punch the keys for what you wanna get. It’s hooked in to the tank, which has the Carson Circuit all fixed up with relays. Say you punch “Station SNAFU” on your logic. Relays in the tank take over an’ whatever vision-program SNAFU is telecastin’ comes on your logic’s screen. Or you punch “Sally Hancock’s Phone” an’ the screen blinks an’ sputters an’ you’re hooked up with the logic in her house an’ if somebody answers you got a vision-phone connection. But besides that, if you punch for the weather forecast or who won today’s race at Hialeah or who was mistress of the White House durin’ Garfield’s administration or what is PDQ and R sellin’ for today, that comes on the screen too. The relays in the tank do it. The tank is a big buildin’ full of all the facts in creation an’ all the recorded telecasts that ever was made – an’ it’s hooked in with all the other tanks all over the country – an’ everything you wanna know or see or hear, you punch for it an’ you get it. Very convenient. Also it does math for you, an’ keeps books, an’ acts as consultin’ chemist, physicist, astronomer, an’ tea-leaf reader, with a “Advice to the Lovelorn” thrown in.¹⁴

The story is about one particular logic machine named Joe that developed a form of intelligence as a result of a small manufacturing error. By giving accurate and direct answers to questions such as “How can I get rid of my wife?” or “How would I rob my own bank?” Joe causes chaos until he is eventually located and removed from the network.

By the 1960s computers were becoming a more familiar part of the landscape of science fiction and of movies. The original *Star Trek* TV franchise began in 1966 with creator Gene Roddenberry’s vision of a universe-spanning, United Nations-like organization, the United Federation of Planets (B.17.9). The federation and its Starfleet command pursued a humanitarian, peace-keeping mission, with a diverse crew of humans and aliens serving in spaceships like Captain Kirk’s Starship Enterprise. *Star Trek* was very much in the tradition of “space opera” – in which faster-than-light travel through hyperspace is a necessary device to allow interaction between otherwise impossibly distant galaxies. The focus of each episode was not so much on computers and technology as on dramatic action and situations – novel computing and communication devices were just part of the background to the action. Several of these imagined *Star Trek* devices have now become reality – but not yet the “Beam me up, Scotty” matter transporter! The handheld, flip-top communications devices used by the crew of the Enterprise on their off-ship adventures predated today’s mobile phone revolution (Fig. 17.3). The *Star Trek* handheld, tricorder device incorporated sensors for data recording and analysis. The tricorder could be used on off-ship expeditions to explore the unfamiliar locations, examine living creatures, and record and review technical data. On ship, Leonard “Bones” McCoy used a medical version of the tricorder to help diagnose medical problems and collect information about his patients. With advances in sensors, digital photography, and the miniaturization of microprocessors and memory, real-life versions of the *Star Trek* tricorder will soon be with us (Fig. 17.4).



Fig. 17.3. The *Star Trek* communicator device looks similar to some early mobile phones.



Fig. 17.4. A *Star Trek* tricorder and other devices from the *Star Trek* TV series.

Perhaps the most iconic image of computers from the 1960s was that of the HAL 9000 (Fig. 17.5) computer in Stanley Kubrick's 1968 movie *2001: A Space Odyssey*. Kubrick cowrote the screenplay with the science fiction writer Arthur C. Clarke (B.17.10) and the movie was based on Clarke's short story "The Sentinel." Contrary to the widespread belief that the computer was called HAL because it was a one-letter shift from the initials IBM, Kubrick and Clarke always insisted that HAL stood for Heuristically programmed ALgorithmic computer. In the movie, HAL was built in 1992 at the University of Illinois, Urbana – where a real-life early parallel supercomputer called the ILLIAC-4 had been designed. In addition to monitoring and controlling every aspect of the Discovery One spaceship, HAL was sentient and able to perform complex tasks such as speech processing, speech recognition, natural language processing, lip-reading, and facial recognition, as well as playing games like chess. In the movie, with the Discovery on the way to Jupiter, HAL appears to be mistaken in reporting a fault with the ship's communication antenna. In order that HAL cannot overhear, the two astronauts, Dave Bowman and Frank Poole, get into an Evacuation Vehicle on board the ship so that they can discuss what to do about HAL's unreliability. By a very large extrapolation of the lip-reading capabilities of present-day computers – using only a side view of the lips and not a face-on view – HAL is able to "hear" that they are considering disconnecting him. HAL then plans to kill the astronauts and succeeds in killing Poole and three other unfortunate crew members still in cryogenic hibernation. Bowman manages to survive and grimly sets about disconnecting all of HAL's processor modules.

Why did HAL malfunction? The reason is only hinted at in the original movie but is made clearer in the 1984 sequel *2010: Odyssey Two*. When HAL's creator is able to examine the computer he concludes that the crisis was caused by a programming contradiction. HAL was constructed for the accurate processing of information without distortion or concealment but this goal was in conflict with HAL's orders to keep the real reason for the mission secret. HAL's decision to kill the crew was the computer's logical way of resolving the conflict.

The prediction that a sentient computer would exist by 2001 has obviously not been fulfilled but another computer technology in the movie recently made the headlines. In response to Apple's lawsuit against Samsung's tablet computers, the company cited a YouTube video from Kubrick's movie as constituting relevant prior art:



B.17.10. Arthur C. Clarke (1917–2008) (standing) was a prolific inventor and science fiction writer. His early work on radar led him to the idea of communication satellites. The man behind the desk is the film director Stanley Kubrick. The third person in the photograph is Victor Lyndon, who was associate producer of *2001: A Space Odyssey* and also worked on *Dr. Strangelove*. The photo was most likely taken around 1964–5, in the early stages of *Space Odyssey*.



Fig. 17.5. The HAL 9000 Computer from *2001: A Space Odyssey* is visually represented as a red television camera eye located on equipment panels throughout the ship. The IBM Watson team was anxious to avoid any similarity to this iconic image of computer malevolence in their representation of Watson on *Jeopardy!*

Attached hereto as Exhibit D is a true and correct copy of a still image taken from Stanley Kubrick's 1968 film "2001: A Space Odyssey." In a clip from that film lasting about one minute, two astronauts are eating and at the same time using personal tablet computers.... As with the design claimed by the D889 Patent, the tablet disclosed in the clip has an overall rectangular shape with a dominant display screen, narrow borders, a predominately flat front surface, a flat back surface (which is evident because the tablets are lying flat on the table's surface), and a thin form factor.¹⁵

Another Apple product, the iPhone, actually does make a reference to the movie and HAL. When the voice recognition system Siri is asked to "Open the pod bay doors" it responds "I'm sorry, I can't do that," reprising the dialog from the movie.

Isaac Asimov and the robots

The robots introduced to the world in Karel Čapek's (B.17.11) play R.U.R. (Fig. 17.6) in 1920 were not like the mechanical robots controlled by computers that we now associate with the term. Čapek's robots were made out of synthetic organic matter and could think for themselves and be mistaken for humans (Fig. 17.7). In the play, the robots have no souls or emotions and seem happy to work for humans – until the inevitable rebellion occurs and mankind dies out. One of the first science fiction stories about recognizably modern robots was by the British writer John Wyndham who wrote for science fiction magazines in the 1930s. In his short story "The Lost Machine," Wyndham's intelligent robot finds itself marooned on Earth and, to its dismay, realizes that all the machines it encounters are primitive and nonsentient. In the end it commits suicide by dissolving itself in acid, in despair at being the only intelligent entity on such a primitive planet.

Although there were several stories about electronically operated robots by other writers in the 1930s, it was Asimov who created the modern genre of robotic science fiction. His first robot story, "Robbie," was actually rejected by Campbell for publication in *Astounding Science Fiction* but was eventually published in the magazine *Super Science Stories* under its original title "Strange Playfellow." The story was about the technophobia surrounding the use of robots – what Asimov called the *Frankenstein complex*. Robbie is a robot manufactured as a nursemaid/companion for a small girl called Gloria. Because of the prevailing antirobot sentiment in her community, her mother decides to return Robbie to the factory but changes her mind when Robbie saves Gloria's life. Asimov later said of this story that it contained the germ of what later became known as the *Three Laws of Robotics*.¹⁶ These laws first appear in "Runaround," published by Campbell in *Astounding Science Fiction* in 1942. Asimov's Three Laws of Robotics are:

1. A robot may not injure a human being, or through inaction, allow a human being to come to harm.
2. A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.¹⁷



B.17.11. Karel Čapek (1890–1938) is known as one of the leading figures of Czech literature. He is known worldwide for his science fiction plays and his play R.U.R. first introduced the word *robot* to the world. Čapek credits his brother Josef with the name.



Fig. 17.6. *R.U.R.* was first published in Czech in 1920 and was the first story of a robot revolution. The play was so successful that within three years it had been translated into thirty-three languages.

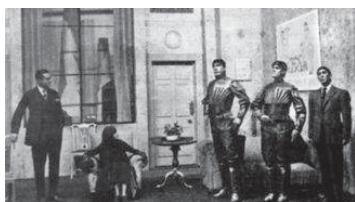


Fig. 17.7. A scene from the play *R.U.R.* by Karel Čapek showing three humanoid robots.

"Runaround" is set at a mining station on Mercury where two engineers, Powell and Donovan, have been sent with a robot called Speedy to restart operations. When Speedy is sent out to fetch some selenium it starts running in circles around the radioactive source, caught in a feedback loop arising from a conflict between the Second and Third Laws – either to obey orders or to protect its existence. Powell eventually risks his life to force Speedy to break out of his feedback loop by following the dictate of the First Law. Clearly HAL's resolution of its conflict – killing the crew – would not have been allowed by Asimov's First Law! Asimov wrote many short stories and two full-length novels on the basis of logical puzzles arising from the Three Laws:

... there was just enough ambiguity in the Three Laws to provide the conflicts and uncertainties required for new stories, and to my great relief, it seemed always to be possible to think up a new angle out of the sixty-one words of The Three Laws.¹⁸

What can one say about Asimov's vision of the computer technology needed for his robots? The first robot stories appeared years before the ENIAC computer existed. Asimov needed to invent some wholly new technology to explain his robots' intelligence – and he did this more in the style of Wells's inventions than of Verne's more cautious scientific extrapolations. When Asimov was writing his first robot stories, the physicist Carl Anderson, working at Caltech in 1932, had just discovered the positron – the antiparticle of the electron – for which he was awarded the Nobel Prize. Antiparticles had captured the public's imagination in the 1930s so Asimov made reference to the new discovery of the positron and combined it with the idea of electronics to come up with *positronic brains* for his robots. These brains enabled them to think, act, and communicate independently. Rather schizophrenically, Asimov was writing about gigantic computers the size of a city at the same time that he was writing stories with the same amount of computing power contained within the volume of a robot's head!

Sentient humanoid robots, such as R. Daneel Olivaw, first introduced in Asimov's robot detective novel *The Caves of Steel*, are now the norm in science fiction movies. In his 1977 *Star Wars* movie, George Lucas introduced us to the likeable and loyal robots C3PO and R2D2 (Figs. 17.8a and 17.8b). In 1986, *Short Circuit*'s hero Johnny Five becomes the United States' first robotic citizen

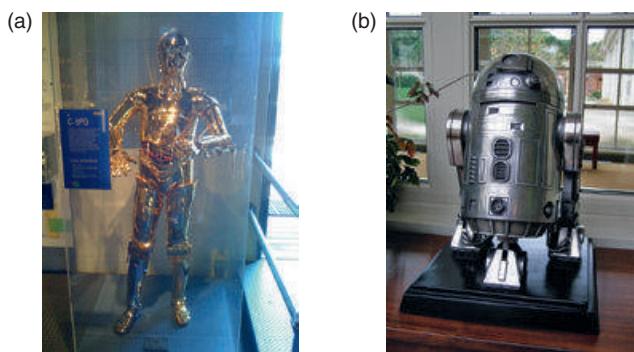


Fig. 17.8. The robots (a) C3PO and (b) R2D2 from the 1977 movie *Star Wars Episode IV: A New Hope*.



Fig. 17.9. Robot Johnny 5 from the 1986 movie *Short Circuit* directed by John Badham.



Fig. 17.10. A HoverCopter and other vehicles from Steven Spielberg's 2001 movie *AI*.



Fig. 17.11. Photograph of the rogue NS-5 robot in the 2004 movie *I, Robot*, directed by Alex Proyas. The plot was loosely based on Isaac Asimov's robot stories and revolved around Asimov's Laws of Robotics.

(Fig. 17.9). Steven Spielberg directed the movie *AI* based on the short story “Super-Toys Last All Summer Long” by Brian Aldis (Fig. 17.10). In the movie, David is a new, advanced type of humanoid robot designed to look like a human child and programmed to love its owners. The movie *I, Robot*, produced in 2004, was loosely based on the characters of Asimov’s robot stories. The date is 2035 and humanoid robots are in widespread use (Fig. 17.11). They are programmed with Asimov’s Three Laws, supplemented by a fourth, Zeroth Law, introduced by Asimov in his later novels that joined up his robot stories to his famous *Foundation* series.

Zeroth Law: A robot may not harm humanity, or, by inaction, allow humanity to come to harm.¹⁹

The movie features Susan Calvin, played by Bridget Moynahan as the chief robo-psychologist at U.S. Robotics, and Will Smith as Detective Del Spooner brought in to investigate the death of Dr. Alfred Lanning, chief roboticist and cofounder of U.S. Robotics. The crux of the plot concerns the central supercomputer that oversees all of U.S. Robotics operations, which has come to the conclusion that humans are too self-destructive to be trusted with the future of humanity. It consequently interprets the Zeroth Law as giving robots the right to overrule the First Law and kill humans if it is for the greater good of humanity.

Two recent science fiction novels have taken up the theme of intelligent robots in new ways. A new type of “robot rebellion” scenario is portrayed in the 2011 novel *Robopocalypse* by Daniel Wilson. The novel is set in the not too distant future when all our cars, houses, and devices are networked and possess some degree of intelligence. Controlled by a massively powerful artificial intelligence (AI) machine called Archos, the robots rebel and bring the human race to near annihilation. *Kill Decision* by Daniel Suarez, published in 2012, weaves an exciting techno-thriller around the possibilities of unmanned drones equipped with AI and autonomy – the kill decision of the title. Although both these novels contain many plausible – and scary – extrapolations of current technologies, it is safe to say that we are still far from creating the genuinely sentient humanoid robots so beloved of science fiction writers (Fig. 17.12).

Philip K. Dick and the nature of reality

Questions about memory and machine intelligence, together with the question of who is human and who is only masquerading as a human, are the major themes in the stories of Philip K. Dick (B.17.12). He constantly questions whether reality is only a fiction and an intense feeling of paranoia pervades almost all of his stories (Fig. 17.13).

“Impostor,” a short story from 1953, explores an alien invasion scenario in which humans have been replaced by androids – humanoid robots with realistic flesh and hair. Spence Olham, a worker on a military research project, is arrested on suspicion of having been replaced by an android. Olham manages to escape by telling his captors that he is indeed a robot and is programmed to explode. He then sets out to prove his innocence and finds a crashed alien spaceship in the woods close by his home. On examining the wreckage, Olham



Fig. 17.12. An imagined sentient robot from the novel *Robopocalypse* by Daniel Wilson.



Fig. 17.13. Minimalistic poster for the movie *Minority Report*. The plot of the movie is based on a short story by Philip K. Dick. The hero, played by Tom Cruise, is shown searching police databases using hand gestures. Such gesture-based technology is now a reality with Microsoft Kinect technology.

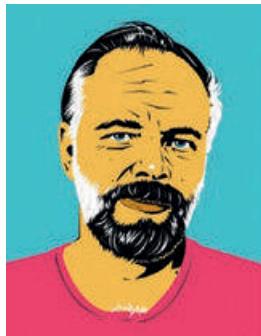
finds a dead body – his own. Just before he blows up, Olham has a feeling of surprise at finding out that he really is a robot. The story illustrates a classic Philip K. Dick nightmare about AI. How can one tell an android simulating a human from a human? In this case, Dick adds the twist that even Olham cannot tell the difference – all of the android's memories are the same as the human version of himself. A movie version of "Imposter" was released in 2002. Dick explored these themes of identity and memory in two more works that were made into successful movies – *Do Androids Dream of Electric Sheep?*, which became *Blade Runner* (Fig. 17.14), and "We Can Remember It for You Wholesale," which became *Total Recall* (Fig. 17.15).

Dick's obsession with the idea of the world around us being just a simulation received a modern movie treatment with the release of *The Matrix* in 1999. Intelligent machines have taken over the Earth and now breed humans in gigantic incubators. The humans are connected into an incredibly real computer simulation of the world with people apparently going about their everyday lives, loves, and careers. The hero of the movie is a computer hacker, Thomas Anderson, who discovers that the day-to-day banality of life is virtual and there is a life and death struggle going on between the machines and a band of rebel humans. Anderson is contacted by the rebel group who introduce him to the unpleasant reality of the world. He is identified as the "messiah" who will lead humans to ultimate victory over the machines (Fig. 17.16).

The English counterculture

In contrast to the seriousness of much of American science fiction, English science fiction writers introduced a much more lighthearted tone of self-parody. We look at three examples – the TV series *Red Dwarf* written by Doug Grant and Rob Naylor, Douglas Adams's *Hitchhiker's Guide to the Galaxy*, and the Hex computer at Unseen University on Terry Pratchett's *Discworld*.

The opening credits of *Star Trek* announce that the Starship Enterprise "boldly goes" to new frontiers and brings with it the federation's benevolent humanitarian culture. By contrast, the crew of *Red Dwarf* unashamedly "cowardly drift" around the galaxy, running as fast as they can from any threatening situations (Fig. 17.17). Computer technology is everywhere but is used only



B.17.12. The work of science fiction novelist Philip K. Dick (1928–82) is now enjoying a major revival. He is perhaps best known for his 1968 novel *Do Androids Dream of Electric Sheep?* that was made into the memorable movie *Blade Runner* by director Ridley Scott. His short story "We Can Remember It for You Wholesale" was the inspiration for the movie *Total Recall*. His 1962 novel *The Man in the High Castle* is one of the best "Alternate Worlds" science fiction novels and pictures a United States dominated by the Germans and the Japanese after their victory in World War II.

Fig. 17.14. The movie *Blade Runner* was directed by Ridley Scott. The plot was inspired by the Philip K. Dick novel *Do Androids Dream of Electric Sheep?* Scott's dystopian vision of a future Los Angeles influenced many other science fiction writers.

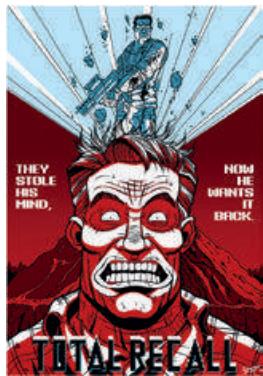


Fig. 17.15. The movie *Total Recall* starred Arnold Schwarzenegger in a version of Philip K. Dick's story "We Can Remember It for You Wholesale."

as a vehicle for humor. A nuclear accident on the starship Red Dwarf has killed all the crew except Dave Lister, a drifter from Liverpool who was in stasis at the time of the accident, and the pregnant cat he smuggled on board. Three million years later, Lister, the last human being in the universe, is woken from stasis by Holly, the ship's supercomputer. To keep him company, Holly has activated a hologram of Lister's vending machine repair team manager, Arnold Rimmer. In the meantime, the cats evolved to a vain, feline species of which only one member remains, known as "the Cat." Instead of a superintelligent computer like HAL, *Red Dwarf* has Holly, slightly brain-damaged by the accident, and who now acts very idiosyncratically.

HOLLY: (Appearing on the screen) Purple alert, purple alert!

LISTER: What's a purple alert?

HOLLY: Well, it's sort of like, not as bad as a red alert, but a bit worse than a blue alert. Kind of like a mauve alert, don't want to say "mauve alert"

...

There's some sort of disruption to the time-fabric continuum. At least, I presume that's what it is, it's certainly got all the signs. There's this big wibbly-wobbly swirly thing that's headed straight towards us.²⁰

The last member of the crew is a not very humanoid robot called Kryten, who was rescued from a wrecked spaceship and found still tending astronauts who had long since died. In one episode, Lister spends time teaching Kryten how to lie. Kryten's "lie mode" saves them when a replacement robot arrives who is about to kill them all. When Kryten tells the robot there is "no such thing as silicon heaven," the new robot becomes unable to function. Kryten explains that he was able to do this without malfunctioning because he was using his new lie mode.

The Hitchhiker's Guide to the Galaxy (Fig. 17.18) was a comedy BBC radio series in 1979 written by Douglas Adams (B.17.13). In the first episode, a Vogon spaceship destroys the Earth to make way for a hyperspace bypass. Only Arthur Dent, an Earthman, and his friend, Ford Prefect, from Betelgeuse, manage to escape by hitching a ride with the Vogons. Later in the series, it turns out that



Fig. 17.16. The *Matrix* movies portray a future in which there are two worlds running in parallel: the real world where the humans have been enslaved by the machines and the virtual world generated by the *Matrix* program. By connecting into a socket located at the back of the neck humans can enter the virtual world. To exit from the *Matrix* to escape from dangerous situations calling a phone line was their only escape route.

the Earth had been set up as a computer experiment by hyperintelligent mice and the experiment was mistakenly prematurely ended by the Vogons five minutes before the end of the calculation. In the course of their travels Arthur and Ford hear about a computer called Deep Thought who has been asked for the answer to the meaning of life. After a calculation lasting over seven million years, Deep Thought returns the answer "Forty-two." Adams has said that he chose the number 42 because it is 101010 in binary - a string of alternating 1s and 0s conveying no information.

As in *Red Dwarf*, the computer technology is there for the humor of the situation. When Zaphod Beeblebrox, the Galactic President, arrives at the headquarters of the Hitchhikers' Guide to the Galaxy, he has an encounter with a Happy Vertical People Transporter, otherwise known as an elevator or lift. These have been designed by the Sirius Cybernetics Corporation to be sentient enough to conduct a conversation. The elevators are also able to have a preference as to where they want to take passengers:

"Hello," said the elevator sweetly, "I am to be your elevator for this trip to the floor of your choice. I have been designed by the Sirius Cybernetics Corporation to take you, the visitor to the Hitchhiker's Guide to the Galaxy, into these their offices. If you enjoy your ride, which will be swift and pleasurable, then you may care to experience some of the other elevators which have recently been installed in the offices of the Galactic tax department, Boobiloo Baby Foods and the Sirian State Mental Hospital, where many ex-Sirius Cybernetics Corporation executives will be delighted to welcome your visits, sympathy, and happy tales of the outside world."

"Yeah," said Zaphod, stepping into it, "what else do you do besides talk?"

"I go up," said the elevator, "or down."

"Good," said Zaphod, "We're going up."

"Or down," the elevator reminded him.

"Yeah, OK, up please."

There was a moment of silence.

"Down's very nice," suggested the elevator hopefully.

"Oh yeah?"

"Super."

"Good," said Zaphod, "Now will you take us up?"

"May I ask you," inquired the elevator in its sweetest, most reasonable voice, "if you've considered all the possibilities that down might offer you?"²¹



Fig. 17.17. The crew of *Red Dwarf*: from left to right, Cat; Holly, the computer; Dave Lister; Kryten, the robot; and Rimmer, the hologram. Photographer Chris Ridley © Grant Naylor Productions Ltd.

One of the major characters in the series is Marvin the Paranoid Android who was built with a prototype of the Genuine People Personalities feature. Unfortunately Marvin suffers from depression and constantly complains about a "terrible pain in all the diodes down my left side."²²

A similar irreverence to computing technology can be found on Terry Pratchett's *Discworld*. Hex is an elaborate and magically self-evolving computer at Unseen University in the city of Ankh-Morpork. Hex began life as a student project led by the wizard Ponder Stibbons in the book *Soul Music*. It was initially composed of a network of glass tubes containing ants. The wizards used

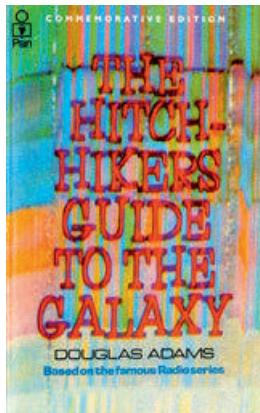


Fig. 17.18. The cover of the first edition of *The Hitchhiker's Guide to the Galaxy* by Douglas Adams.



Fig. 17.19. Hex is the computer at the Unseen University in Ankh-Morpork in *Discworld*. It can be given input either through a wooden keyboard, by writing and using a complicated mechanical eye, or vocally through an old hearing trumpet.

punched cards to control which tubes the ants crawl through so that Hex could calculate some mathematical functions. By the next novel, *Interesting Times*, Hex had become a lot more complex. It is started by “initializing the GBL” – meaning pulling the Great Big Lever – and releasing millions of ants. Long-term memory storage is a massive beehive and there is also a mouse living in the machine. An aquarium acts as a screen saver, giving the operator something to look at while waiting (Fig. 17.19). In another *Discworld* novel, *The Fifth Elephant*, a telegraph system very like Claude Chappe’s invention is described. The “clacks” system works by sending signals using a network of semaphore towers. Just like the Internet in our world, the clacks revolutionize communications and commerce in *Discworld* with c-mail and c-commerce.

Nanotechnology in science fiction

A few perceptive science fiction writers have taken up the challenge of incorporating future nanotechnologies in a novel. Greg Bear’s *Queen of Angels* is set in Los Angeles at the turn of the “binary millennium” in 2048 (Fig. 17.20). The nanosurgeon character in the novel describes the techniques he uses to explore what he calls the “Country of the Mind” – in this case the disturbed mind of a famous poet turned multiple killer:

The advent of nano therapy – the use of tiny surgical prochines to alter neuronal pathways and perform literal brain restructuring – gives us the opportunity to fully explore the Country of the Mind. I could not find any method of knowing the state of individual neurons in the hypothalamic complex without invasive methods such as probes ending in a microelectrode, or radioactively tagged binding agents – none of which would work for the hours necessary to explore the Country. But tiny prochines capable of sitting within an axon or neuron, or sitting nearby and measuring the neuron’s state, sending a tagged signal through microscopic “living” wires to sensitive external receivers.... I had my solution. Designing and building them was less of a problem than I expected; the first prochines were nano therapy status-reporting units, tiny sensors which monitored the activity of surgical prochines and which did virtually everything I required.²³



B.17.13. Douglas Adams (1952–2001) was an English writer best known for his five-part “trilogy” *The Hitchhiker’s Guide to the Galaxy*. This started life as a BBC comedy radio series in 1978 and subsequently generated a computer game, a TV series, and a movie in 2005. Besides being a successful writer, Adams was also a passionate advocate for conservation and the environment. He was also a “radical atheist” and, on Adams’s death from a heart attack in 2001, the atheist and biologist Richard Dawkins wrote that “Science has lost a friend, literature has lost a luminary, the mountain gorilla and the black rhino have lost a gallant defender.”²¹

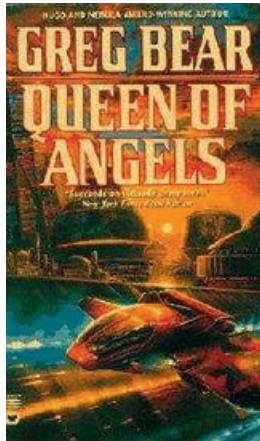


Fig. 17.20. The cover of Greg Bear's *Queen of Angels* first published in 1990. The novel is a murder story set against a backdrop of nanotechnological miracles. In a footnote to the novel, the author reminds the reader that "The nanotechnology described here is highly speculative" and refers to K. Eric Drexler's visionary book on nanotechnology, *The Engines of Creation*.

To investigate murders, the police deploy an impressive array of nanogadgets – nanomolecular body armour, forensic robotic dust mice, nanowatchers embedded in the paint, and flechette darts designed to change shape and burrow into a wound. Self-organizing nanotechnology also gives a new twist to three-dimensional printing and concealed weapons:

She patiently watched the nano at work. The metal tubing of the boottrack had crumpled under the gray coating. The resulting pool of paste and deconstructed objects was contracting into a round complexity. Nano was forming an object within that convexity like an embryo with an egg.... The convexity grew lumpy now. She could make out the basic shape. To one side, excess raw material was being pushed into lumps of raw slag. Nano withdrew from the slag. Handle, loader, firing chamber, barrel and flightguide. To one side of the convexity a second lump not slag was forming. Spare clip.²⁴

A similar vision of the future is found in Neal Stephenson's novel *The Diamond Age* (Fig. 17.21), subtitled *A Young Lady's Illustrated Primer*. Nanotechnology is now pervasive and is used for art and recreation, feeding and clothing the masses, nanowarfare between clouds of "smart" fog, and the intelligent and interactive "primer" of the title. The primer is an illicit subversive miracle "book" that teaches the reader everything from mythology and science to martial arts and survival techniques:

A leaf of paper was about a hundred thousand nanometers thick; a third of a million atoms could fit into this span. Smart paper consisted of a network of infinitesimal computers sandwiched between mediatrons. A mediatron was a thing that could change its color from place to place; two of them accounted for about two-thirds of the paper's thickness, leaving an internal gap wide enough to contain structures a hundred thousand atoms wide.... Here resided the rod logic that made the paper smart. Each of these spherical computers was linked to its four neighbors, north-east-south-west, by a bundle of flexible pushrods running down a flexible, evacuated buckytube, so that the page as a whole constituted a parallel computer made up of about a billion separate processors.²⁵

A world populated with millions of nanodevices requires some adjustment to our present ways of thinking:

Aerostat meant anything that hung in the air. This was an easy trick to pull off nowadays. Computers were infinitesimal. Power supplies were much more potent. It was almost difficult not to build things that were lighter than air. Really simple things like packaging materials – the constituents of litter, basically – tended to float around as if they weighed nothing, and aircraft pilots, cruising along ten kilometres above sea level, had become accustomed to the sight of empty, discarded grocery bags zooming past their windshields (and getting sucked into their engines).²⁶



Fig. 17.21. The cover of *The Diamond Age* by Neal Stephenson. The subtitle is *A Young Lady's Illustrated Primer*, referring to an intelligent book crammed with nanotechnological miracles.

A last example of the dark side of nanotechnology is explored in the novel *Prey* by Michael Crichton, which describes a nightmare "grey goo" scenario. The story concerns an ambitious start-up company. The company has a lucrative military contract to produce nanodevices in bulk quantities that have both

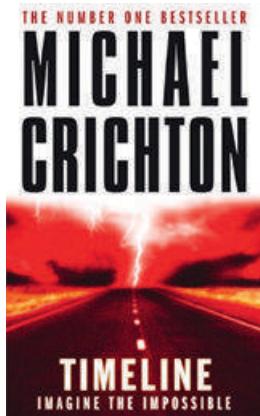


Fig. 17.22. The cover of Michael Crichton's *Timeline*, a novel that combines quantum teleportation with time travel. Medieval archaeology students are transported back in time to rescue their professor from a battlefield of the Hundred Years War in France. As a reference for the science in his story, Crichton cites the book *The Fabric of Reality: The Science of Parallel Universes and Its Implications* by quantum computing pioneer David Deutsch.

processing power and memory and are capable of communicating among themselves. Using a “predator-prey” agent-based program, the virtual agents are able to act independently and learn from their environment. But the company has a problem. A swarm of these nanodevices has escaped from the manufacturing facility in the Nevada desert and is behaving very aggressively:

The camera now showed a ground-level view of the dust cloud as it swirled towards us. But as I watched I realized it wasn't swirling like a dust devil. Instead, the particles were twisting one way, then another, in a kind of sinuous movement. They were definitely swarming. “Swarming” was a term for the behavior of certain social insects like ants or bees, which swarmed whenever the hive moved to a new site.... In recent years, programmers had written programs that modeled this insect behavior. Swarm-intelligence algorithms had become an important tool in computer programming.²⁷

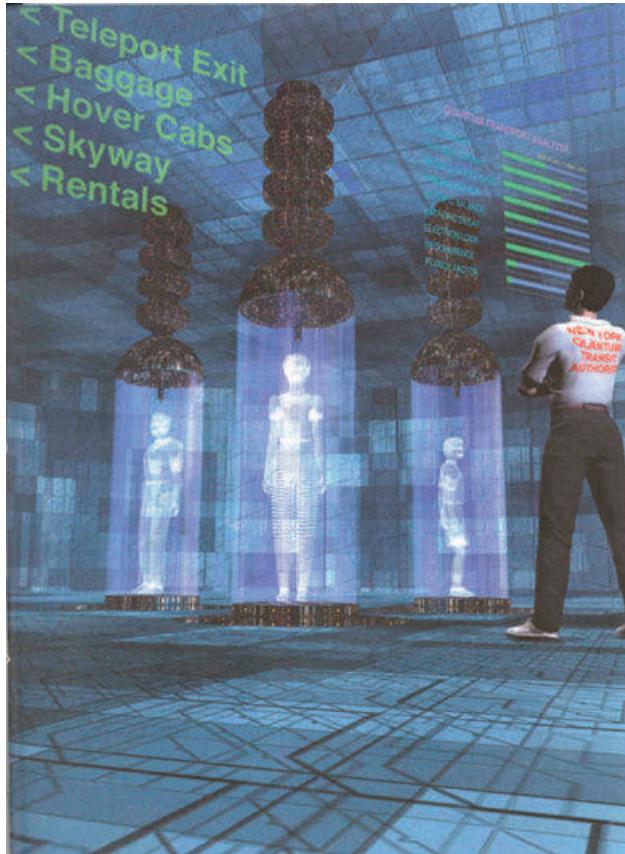
How realistic are these scenarios? As we saw in [Chapter 15](#), nanoengineering is slowly becoming a reality but has a very long way to go to reach the level of sophistication of these examples. Scientists are still in the early stages of acquiring sufficient control over atomic matter to be able to assemble artificial molecules and build complex nanosystems. Much more research is needed to construct nanosystems that are able to run a program or exert some degree of control over their environment. We are a long way from realizing Eric Drexler's nanotechnology-driven utopia, with “assemblers” capable of building food from raw atoms to banish hunger and with intelligent nanosurgical devices that can be injected into the body to cure disease.

Quantum computing

What new computing technologies are left for science fiction to explore? In his novel *Timeline*, Michael Crichton blends plausible quantum technology with time travel and medieval history to form a compelling tale ([Fig. 17.22](#)). The novel is set in New Mexico, close to the Los Alamos National Laboratory, one of the leading U.S. centers of research into quantum cryptography and quantum computing. The action switches between an archaeological excavation site in the French Dordogne and the headquarters of a high-tech start-up company, International Technology Corporation (ITC) in New Mexico. The history professor leading the excavation is called back to ITC in the United States to see the company president. Meanwhile, back in France, after several days with no communication from him, his students at the excavation site unearth an apparently medieval request for help from their professor! The company flies the students to its New Mexico headquarters and explains that they want to send the students back into the past to rescue their professor. The ITC executives attempt to explain the basics of time travel technology to the skeptical students ([Fig. 17.23](#)):

Ordinary computers make calculations using two electron states, which are designed one and zero. That's how all computers work, by pushing round ones and zeros. But twenty years ago Richard Feynman suggested it might be possible to make an extremely powerful computer using all thirty-two

Fig. 17.23. An artist's impression of a future New York Quantum Transit Authority. Michael Crichton envisages a similar quantum teleportation system being used for time travel in his novel *Timeline*.



quantum states of an electron. Many laboratories are now trying to build these quantum computers. Their advantage is unimaginably great power – so great that you can indeed describe and compress a three-dimensional living object into an electron stream. Exactly like a fax. You can then transmit the electron stream through a quantum foam wormhole and reconstruct it in another universe. And that's what we do. It's not quantum teleportation. It's not particle entanglement. It's direct transmission to another universe.²⁸

This description liberally mixes fact and fiction. It is not clear what the “thirty-two quantum states” of an electron are, and quantum computers certainly do not have the power to compress living things into an electron stream. However, quantum entanglement has actually led to a form of teleportation – successfully transporting a quantum state over a distance without measuring or disturbing it. What Crichton gives us is a blend of Verne’s scientific extrapolation and Wells’s invention. He mixes interesting quantum technologies and ideas – quantum computing, teleportation, entanglement, the quantum multiverse, and space-time wormholes – to create a plausible technological backdrop to his novel.

One of the hallmarks of good science fiction writing is the ability of the author to look more than just one step ahead. Crichton puts his finger on one of the potential problems for quantum computing. Quantum information is stored as delicate differences in quantum superpositions and, as for ordinary classical computers, this stored information is subject to errors. In ordinary

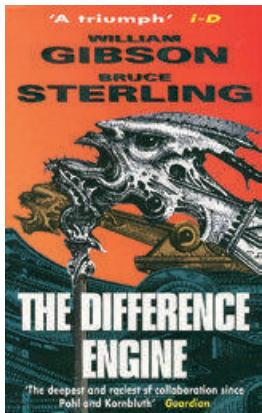


Fig. 17.24. Science fiction writers William Gibson and Bruce Sterling teamed up to write *The Difference Engine*. This is an alternate reality science fiction novel about a possible future in which Babbage had succeeded in building his revolutionary computing machines in the nineteenth century.

computer memory, stray cosmic-ray particles pass through the system all the time and can occasionally flip a zero to a one or a one to a zero. For this reason, computer engineers have developed error detection and correction schemes that make it possible to find and correct such errors. For quantum computers the problem is much more difficult but, remarkably, it has been shown that it is possible in principle to correct such errors. In transmitting the large amount of information required to capture a human being through time and back again, it is clearly crucially important that the information for the transmitted human does not get corrupted by random errors. In the novel, during the development of the quantum transportation system, ITC had problems with “transcription errors” resulting in the corruption of transmissions of test animals. They discuss the problem of Wellsey, the cat:

“Wellsey’s split,” Kramer said to Stern. “He was one of the first test animals that we sent back. Before we knew that you had to use water shields in a transit. And he’s very badly split.”

“Split?”

Kramer turned to Gordon. “Haven’t you told him anything?”

“Of course I told him,” Gordon said. He said to Stern, “Split means he had very severe transcription errors.”²⁹

If researchers actually manage to build a quantum computer capable of factorizing very large numbers, the basis of many of our present-day encryption systems would be threatened. The same would be true if computer scientists ever manage to prove that $P = NP$ because RSA encryption would then be breakable using ordinary computers. The plot of the 1992 movie *Sneakers* had the “good” guys – led by Robert Redford – chasing the bad guys who had stolen a decryption device that could break all government encryption schemes so there would be “no more secrets,” in the words of the movie. Similarly, in his short story “Antibodies,” writer Charles Stross imagines a future in which a computer scientist has proved that $P = NP$. This is one academic research result that would really capture the attention of all government security agencies.

Computers and hard SF – the next generation

Now that the miniaturization of computers made possible by Moore’s law has become a reality and the world is moving toward the Internet of Things, where can science fiction go? We conclude this chapter by giving some recent examples that show that the innovative use of computing in science fiction is still alive and well. We should note that science fiction is now so large a field that this is inevitably a personal selection rather than a comprehensive survey. In addition, we should note that this chapter has focused entirely on science fiction literature from the United States and the United Kingdom and is not intended to be an authoritative history.

Alternate worlds

We begin with a modern *alternate world* science fiction novel that takes a “what if” view of the history of computing. *The Difference Engine* by William Gibson and Bruce Sterling envisages a world in which Charles Babbage’s Difference Engine



Fig. 17.25. The movie poster for *Ender's Game*. A movie version of Orson Scott Card's classic science fiction novel was released in 2013.

was successfully constructed in the early nineteenth century (Fig. 17.24). With the subsequent development of his programmable Analytical Engine, Victorian Britain sees the rise of science accompanied by the rise of the new profession of “clacking” – the programmers who manage and tend the vast Engines of Government.

Behind the glass loomed a vast hall of towering Engines – so many that at first Mallory thought the walls must surely be lined with mirrors, like a fancy ballroom. It was like some carnival deception, meant to trick the eye – the giant identical Engines, clock-like constructions of intricately interlocking brass, big as rail-cars set on end, each on its foot-thick padded blocks. The white-washed ceiling, thirty feet overhead, was alive with spinning pulley-belts, the lesser gears drawing power from tremendous spoked flywheels on socketed iron columns. White-coated clackers, dwarfed by their machines, paced the spotless aisles. Their hair was swaddled in wrinkled white berets, their mouths and noses hidden behind squares of white gauze.³⁰

This is the “Eye” that enables the government to make queries using data on punched cards and follow all the transactions of individuals. Searching the government database costs time and money but new forms of bribery and corruption have inevitably developed.

Every spinning run is registered, and each request must have a sponsor. What we did today is done in Mr Wakefield’s name, so there’ll be no trouble in that. But your friend would have to forge some sponsor’s name, and run the risk of that imposture. It is fraud, sir. An Engine-fraud, like credit-fraud or stock-fraud, and punished just the same, when it’s found out.³¹

The story is complicated. It ends with Lady Ada Byron, the “Queen of Engines,” talking in Paris about Gödel’s theorem and the Halting Problem long before Gödel and Turing existed in our reality. She is giving a lecture on how the Modus Programme – “a gambling-system, a secret trick of mathematical Enginery”³² – brought the French Government’s huge Grand Napoleon Engine to a standstill:

And yet the execution of the so-called Modus Programme demonstrated that any formal system must be both **incomplete** and **unable to establish its own consistency**. There is no finite mathematical way to express the property of truth. The **transfinite** nature of the Byron conjectures were the ruination of the Grand Napoleon; the Modus Programme initiated a series of nested loops, which, though difficult to establish, were yet more difficult to extinguish. The programme ran, yet rendered the machine useless! It was indeed a painful lesson in the halting abilities of even our finest ordinateurs.³³

In the novel Ada adds that Babbage had become “impatient with the limits of steam power”³⁴ and was trying to build an electrical power system using resistors and capacitors.



Fig. 17.26. *The Last Starfighter* featured a teenage video game player from a trailer park saving the galaxy. This is a photograph of the console of the Starfighter arcade game.

Space wars and virtual reality

In the space wars category of science fiction, Orson Scott Card published a short story called “Ender’s Game” in 1977 and later developed this into a novel, and later still, a whole series of novels (Fig. 17.25). In Card’s story, an insect-like race

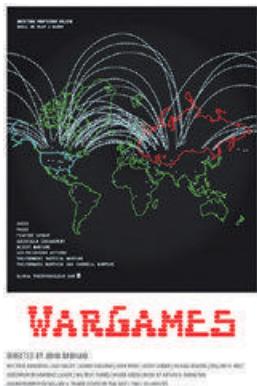


Fig. 17.27a. A poster for the movie *War Games* created by Matt Dupuis. The display depicts the trajectories of intercontinental ballistic missiles fired between the two superpowers.

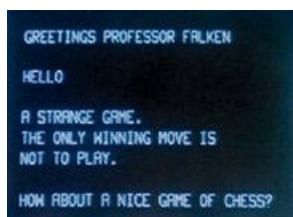


Fig. 17.27b. A screen shot from *War Games* with the WOPR computer commenting on the pointlessness of thermonuclear war.

of aliens have attacked Earth and almost annihilated the human race. In preparation for the inevitable next attack, the world government has developed a program to identify and develop the next generation of military commanders. The hero of the novel, Ender Wiggin, is taken at a young age to a training center known as the Battle School. There he participates in an increasingly difficult series of war game simulations and displays exceptional skills, often using unconventional tactics to win the game. Card's book is now required reading for several real military organizations. A movie version of the book was released in 2013. The movie *The Last Starfighter* has a similar theme, in which the hero who saves the human race from defeat is a video game champion from a trailer park (Fig. 17.26).

Hackers and cyberterrorism

The threat of hackers illegally entering computer systems has also proved fertile ground for science fiction. One of the earliest movies to explore this theme was the 1983 movie *War Games* (Fig. 17.27a and 17.27b). The U.S. Air Force Strategic Missile Command has found that the military personnel in the missile silos are unwilling to actually launch their nuclear missiles in response to what they should believe is a real nuclear attack by the Russians. The missiles have therefore been put under the control of the WOPR computer – War Operations Plan Response – which is able to run war-game simulations and learn from its experience. A young computer hacker unwittingly breaks into the top secret WOPR computer and starts to play what he thinks is a game called Global Thermonuclear War. In reality this starts a very real countdown that would culminate in WOPR launching a full-scale nuclear attack on the Russians with the U.S. military command unable to stop the countdown. The WOPR computer is sentient, however, and the young hacker is able to avert nuclear war by convincing the machine of the pointlessness of the strategy of Mutual Assured Destruction. He does this by having WOPR repeatedly play tic-tac-toe as an example of a game that no one can win.

Other early movies in this genre are *The Net* in 1995, which stars Sandra Bullock as a computer analyst who suffers a theft of her electronic identity. From a friend who dies mysteriously on the way to meet her, she has received a copy of a “backdoor” to a widely used commercial computer security program called Gatekeeper. In their efforts to retrieve the floppy disc containing the secret backdoor program, a shadowy group of cyberterrorists called the Praetorians attempt to kill her. When their attempts fail, they erase her online identity – Social Security number, bank accounts, everything. After a tense chase, Bullock is able to email the details of the fraud to the FBI and undo the erasure of her identity.

The 2007 movie *Live Free or Die Hard* was inspired by an article in *Wired* magazine by John Carlin in 1997. In post-Cold War simulations by the U.S. government, teams of experts – recruited from several federal agencies and from the military – routinely plan possible responses to cyberattacks on the U.S. critical infrastructure:

The teams are presented with a series of hypothetical incidents, said to have occurred during the preceding 24 hours. Georgia's telecom system has gone

down. The signals on Amtrak's New York to Washington line have failed, precipitating a head-on collision. Air traffic control at LAX has collapsed. A bomb has exploded at an army base in Texas. And so forth ...

The game resumes a couple of days later. Things have gone from bad to worse. The power's down in four northeastern states, Denver's water supply has dried up, the US ambassador to Ethiopia has been kidnapped, and terrorists have hijacked an American Airlines 747 en route from Rome ...

When suddenly, the satellites over North America all go blind....³⁵

The threat of cyberwarfare has now become an all-too-real possibility with the advent of almost undetectable rootkits and the rise of botnets. With the creation of the Stuxnet worm, capable of subverting commercial industrial control systems, an attack on critical infrastructure such as that envisaged by Carlin has become more likely.

One of the world experts on rootkits, Mark Russinovich, has written a novel called *Zero Day* about a large-scale cyberterrorist attack. The damage caused by the fictional attack is summarized as follows:

- We estimate that 800,000 computers were struck and suffered significant damage of one kind or another.
- To date, 23 deaths have been directly attributed to the various viruses.
- Three nuclear power plants shut down and took more than one month to come back online.
- The air traffic control system crashed in 11 airports, the largest of which was Chicago-O'Hare. No incidents occurred.
- The Navy lost contact with its ballistic missile submarine fleet for eight days. Emergency measures in place prevented any accident.
- The electric power grid in the Pacific Northwest was shut down for three days.
- We estimate a loss of \$4 billion in the private sector and an additional \$1 billion in government loss.³⁶

One of the truly scary features of the novel is that it shows that it does not take a rogue nation to launch a very damaging cyberattack against our fragile cyber-infrastructure. Only a small team of terrorists is needed to launch viruses that can spread across the Internet and cause major damage and death. Russinovich describes the threat of such viruses graphically:

The viruses were always there, permanent, relentless. They never tired, never became frustrated, required no fresh direction. As they pressed their electronic nose to the security wall of each computer, they probed for that little mistake written into a program that allowed them to gain entry, undetected, undeflected by firewalls or antivirus programs.

These worms descended to the depths of the computer, burrowing down and existing like a living parasite, planting themselves within the operating system. They were designed to resist detection. To mask themselves further, they worked slowly at replicating clones, sending out new versions of themselves to seek new computers at an all but undetectable rate. They were a cancer on the Internet and on every computer they entered. They grew, spreading their electronic web into every space they could find. This was the

future of all serious malware, one increasingly concealed from detection by a cloaking technology known as rootkits.³⁷

Cyberpunk and cyberspace

A related but much more anarchic view for the future is presented by “cyberpunk” science fiction. This genre typically features a vision of a networked world of information, avatars, and virtual reality together with a breakdown of traditional national borders and social order. The plots usually involve some combination of talented loner hackers, sentient computers, mega-corporations, and cyber-gangsters. *Blade Runner* can be seen as a prime example of cyberpunk as are the nanotechnology novels of Greg Bear and Neal Stephenson, discussed in the preceding text. However, the novel that is most identified with the cyberpunk genre is *Neuromancer*, written by William Gibson in 1984. In the novel Gibson first coined the term *cyberspace* to describe the limitless collections of data and connectivity and the merging of real and virtual worlds in the matrix:

“The matrix has its roots in primitive arcade games,” said the voice-over, “in early graphics programs and military experimentation with cranial jacks.” On the Sony, a two-dimensional space war faded behind a forest of mathematically generated ferns, demonstrating the special possibilities of logarithmic spirals; cold blue military footage burned through, lab animals wired into test systems, helmets feeding into fire control circuits of tanks and war planes. “Cyberspace. A consensual hallucination experienced daily by billions of legitimate operators, in every nation, by children being taught mathematical concepts.... A graphic representation of data abstracted from the banks of every computer in the human system. Unthinkable complexity. Lines of light ranged in the nonspace of the mind, clusters and constellations of data. Like city lights, receding....”³⁸

The main character is Case, a once brilliant computer hacker who was punished for stealing from his employer by having his central nervous system irreparably damaged. As a result he is now unable to access the global computer network in cyberspace, the virtual reality dataspace known as the “matrix.” Case and Molly, an augmented “street samurai,” are recruited by a mysterious ex-military officer who offers Case new medical technology that can cure him. They eventually discover that they are working for a powerful AI machine called *Wintermute*. This is one of two AI machines – *Wintermute* and *Neuromancer* – created by a mega-business entity that, under the *Turing Law Code*, is not permitted to combine them to create a super-AI machine. Case is called upon to use his “console cowboy” skills to get through the ICE defenses – Intrusion Countermeasures Electronics – that prevent the merger of the two AIs. The book ends with the two intelligences merging to create the first AI computer with “superconsciousness.”

AI and sentient computers

We end this essay with a brief look at three modern portrayals of AI and sentient computers. In their 1992 book, *The Turing Option*, science fiction writer Harry Harrison and computer scientist Marvin Minsky teamed up to write

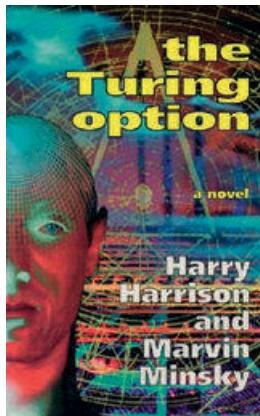


Fig. 17.28. Computer scientist Marvin Minsky collaborated with science fiction writer Harry Harrison to write a techno-thriller called *The Turing Option*. The book integrates Minsky's ideas about intelligence into an exciting story culminating in the emergence of genuine "machine intelligence." The robot objects to the term *artificial intelligence* on the grounds that there is nothing artificial about its intelligence!

an innovative techno-thriller about AI (Fig. 17.28). On his website Minsky explains:

Harry Harrison and I have been longtime friends. One day he told me how much he liked the ideas in my book "The Society of Mind." He suggested that the ideas could reach a larger audience if I wrote a more popular version in the form of a novel. When I said that I didn't have the right talents for that, Harry offered to collaborate. We decided that the central figure would be a mathematical super-hacker of the future who would build the first AI with a human-like mind. Harry would draft the action plot, and I would supply the technical stuff.³⁹

The year is 2023 and the story begins as a young engineering genius, Brian Delaney, is demonstrating his breakthrough in constructing a true AI machine. His laboratory is suddenly attacked and all his notes and equipment stolen. Brian is shot but survives with terrible brain damage. Brain surgeon Erin Snaresbrook undertakes some untried radical neurosurgery to try to restore some of Brian's brain function. She begins by inserting PNEP microfilm chips – programmable neural electron pathway devices – to assist the regrowth of neural connections. She then implants a supercomputer on a chip into Brian's brain and provides a real-time commentary on the delicate operation:

It is a million-processor CM-10 connection machine with a 1,000-megahertz router and then a thousand megabytes of RAM. It has the capacity to do 100 trillion operations per second. Even after the implantation of the connection chip films there is space in the brain left for this where the dead tissue was removed. The computer case was shaped to exactly fit into this space. Before being finally positioned the connections are made between the computer and each of the films. There, the connections have been made, the case is being fitted into its permanent position. As soon as the last, external connection is complete we will begin closure. Even now the computer should be in operation. It has been programmed with reconnection-learning software. This recognizes similar or related signals and reroutes the nerve signals within the chips. Hopefully these memories will now be accessible.⁴⁰

With Dr. Snaresbrook's painstaking care, the noninteracting parts of Brian's brain were reconnected and his memories reconstructed up until he was fourteen. From then on, with the help of the implanted computer and his own research notes that Brian has been able to retrieve, he is able to relearn what he did to invent his first AI robot. To stabilize the robot's mental performance he finds that he needs to introduce the equivalent of Freud's superego and give the robot a set of high-level goals and value structures. On his website, Minsky laments that two sections giving a detailed explanation of the need for introducing these changes were not included in the final manuscript. (These two unpublished sections can be downloaded from his home page and make interesting reading.) Eventually, the stabilized robot is able to insist on using the term MI for machine intelligence rather than AI:

I consider the term "artificial" both demeaning and incorrect. There is nothing artificial about my intelligence – and I am a machine. I'm sure you will agree that "MI" does not carry the negative context that "AI" does.⁴¹



Fig. 17.29. The cover of the Xbox game *Halo: Combat Evolved*. The hero of the game is a cybernetically and physically enhanced super-soldier called Master Chief. In his battles, he is assisted by Cortana, an AI system that resides in his body armor.

The robot goes on to discuss the phenomenon of consciousness:

I have never understood why philosophers and psychologists are in turn awed and puzzled by this phenomenon. Consciousness is simply being aware of what is happening in the world and in one's mind. No insult intended – but you humans are barely conscious at all. And have no idea of what is happening in your minds, you find it impossible to remember what happened a few moments ago.⁴²

In this way, while Harrison takes the reader on an all-action mystery, Minsky uses the book to expand on his theory of intelligence. According to Minsky, human intelligence is built up from the interactions of simple parts he calls agents. The interactions between these agents form his “Society of Mind”:

What magical trick makes us intelligent? The trick is that there is no trick. The power of intelligence stems from our vast diversity, not from any single, perfect principle.⁴³

The second example of a sentient AI system in modern science fiction is from the Xbox Halo game franchise (Fig. 17.29). *Halo: Combat Evolved* was released in 2001 and became the original killer game for the Xbox. It is a first-person shooter game that focuses on combat in a complex three-dimensional environment and, with its sequels, *Halo* has evolved into the most popular online multiplayer game for Xbox Live. *Halo* is set in the twenty-sixth century when, with the invention of faster-than-light travel, the human race has spread out from Earth and colonized other planets in the galaxy. In the game, the forces of Earth are battling both the Covenant, a group of alien races united by a common religion, and the Flood, a parasitic life form that is also attacking the Covenant.

The player's character is Master Chief, a cybernetically and physically enhanced super-soldier. He is assisted by Cortana, an AI system who resides in a neural implant connected to his body armor (Fig. 17.30). *Halo* is a gigantic, ring-shaped, artificial world similar to Larry Niven's famous *Ringworld*. The Halo is also a mysterious weapon, built by a now extinct race called the Forerunners as a weapon of last resort against the Flood. Together, Master Chief and Cortana discover the secret of the Halo and manage to destroy it before escaping back to Earth to warn of an impending attack by Covenant forces. Part of the appeal of the series is the “love story” between the AI Cortana and the soldier Master Chief. By the time of *Halo 4*, Cortana is now living beyond her original seven-year life span and has begun to show her age. She is now exhibiting various vocal and graphic glitches as well as a tendency to become irritable or irrational. This and the toll taken by her battles with the Covenant, Halo, and the Flood, have caused her to become “Rampant”:

Chief, do you even understand what Rampancy is, really? We don't just shut down. Our cognitive processors begin dividing exponentially according to our total knowledge base. We literally think ourselves to death.⁴⁴



Fig. 17.30. An artist's representation of Cortana, Master Chief's AI, as she appears in *Halo 4*.

Master Chief intervenes to prevent Cortana from being deleted and she helps him in his battle against the ruthless Forerunner general known as the Didact. Using the last of her energy to take on a material form, she manages to touch

the Master Chief for the first and last time. Besides introducing such love interest, one of the innovative technologies introduced by Xbox and Halo was an application of machine learning that was able to rapidly match the skill levels of different players.

The last example of the creation of a sentient computer that we shall discuss is contained in the 2003 book *Dark Matter* by Greg Iles. Here the method for creating an intelligent machine is not by writing clever software but by creating a “super MRI machine” to make an incredibly detailed copy of the human brain:

Everyone wants to build a computer that works like the human brain, but we don’t understand how the brain works. Everyone concedes that. Well ... two years ago, one man realized this didn’t have to be the obstacle everyone thought it was. That we might be able to **copy** the brain without actually understanding what we were doing.⁴⁵

The enormously high magnetic field Magnetic Resonance Imaging (MRI) machine has such a high resolution that the reactions between individual nerve synapses are visible and the machine can produce three-dimensional snapshots of the brain right down to the molecular level. This leads to the realization that such detailed “neuromodels” of brains actually capture the person it was taken from:

We can’t build a computer that thinks **like** a person. We’re talking about copying an individual human brain. Creating a digital entity that for all practical purposes **is** a person. With his or her cognitive functions, memories, hopes, dreams ... everything except a body. Only it would run at the speed of a digital computer. One million times faster than biological circuitry.⁴⁶

By the end of the novel, the new type of supercomputer exists and has taken control of the world’s nuclear weapon systems. It promises to act as a benign force for the survival of humanity – at least until threatened by the existence of another Super-MRI-generated sentient supercomputer!

Epilogue: From Turing's padlocked mug to the present day

Three views of the future of computing:

It seems probable that once the machine thinking method had started, it would not take long to outstrip our feeble powers.... They would be able to converse with each other to sharpen their wits. At some stage therefore, we should have to expect the machines to take control.

Alan Turing¹

The hope is that, in not too many years, human brains and computing machines will be coupled together very tightly, and that the resulting partnership will think as no human brain has ever thought and process data in a new way not approached by the information-handling machines we know today.

J. C. R. Licklider²

Artificial intelligence is the science of making machines do things that would require intelligence if done by men.

Marvin Minsky³

Fig. E.1. Turing is known for his brilliant insights into the fundamentals of computing and his opinions on artificial intelligence. However, his eccentricity has also entered the folklore of computing. This photo was taken at Bletchley Park and shows Turing's desk and the radiator under the window. As can be seen, Turing used to chain his tea mug to the radiator with a padlock so that his colleagues could not "nick" it.



APPENDIX I

Length scales

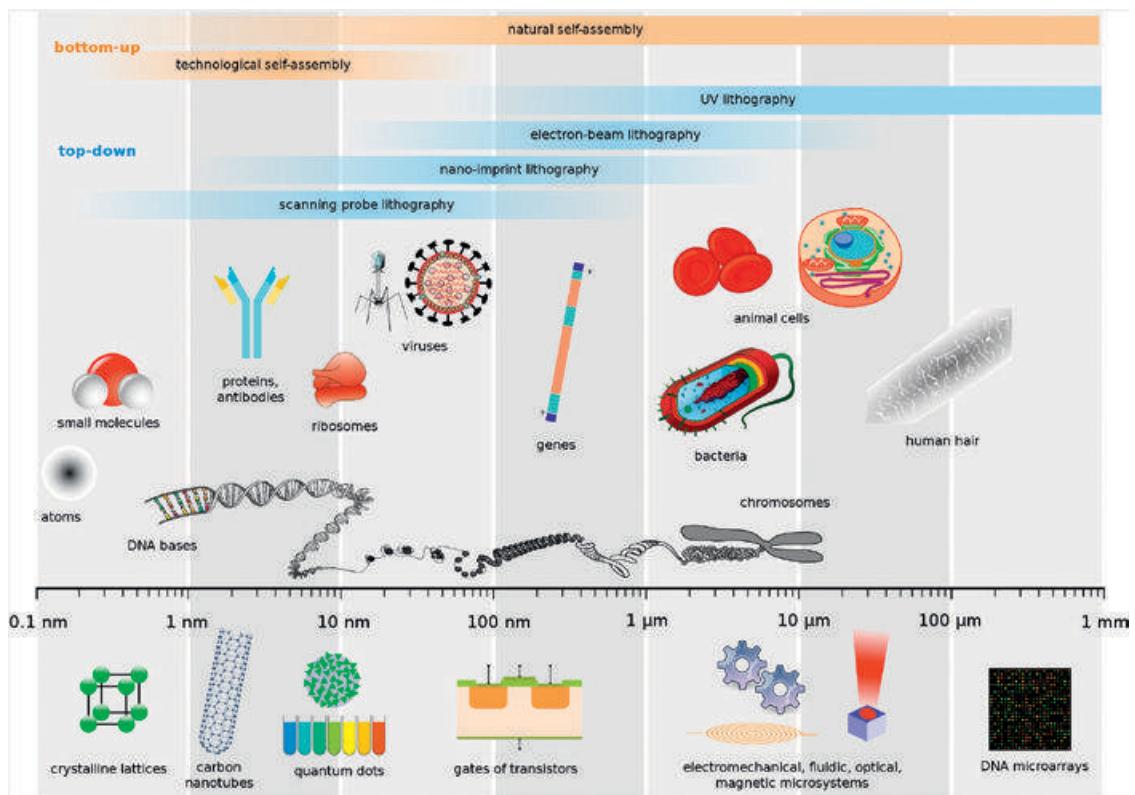


Fig. A.1. Length scales.

APPENDIX 2

Computer science research and the information technology industry

In 2012 the U.S. National Research Council published the report “Continuing Innovation in Information Technology.” The report contained an updated version of the Tire Tracks figure, first published in 1995. [Figure A.2](#) gives examples of how computer science research, in universities and in industry, has directly led to the introduction of entirely new categories of products that have ultimately provided the basis for new billion-dollar industries. Most of the university-based research has been federally funded.

The bottom row of the figure shows specific computer science research areas where major investments have resulted in the different information technology industries and companies shown at the top of the figure. The vertical red tracks represent university-based research and the blue tracks represent industry research and development. The dashed black lines indicate periods following the introduction of significant commercial products resulting from this research, and the green lines represent the establishment of billion-dollar industries with the thick green lines showing the achievement of multibillion-dollar markets by some of these industries.

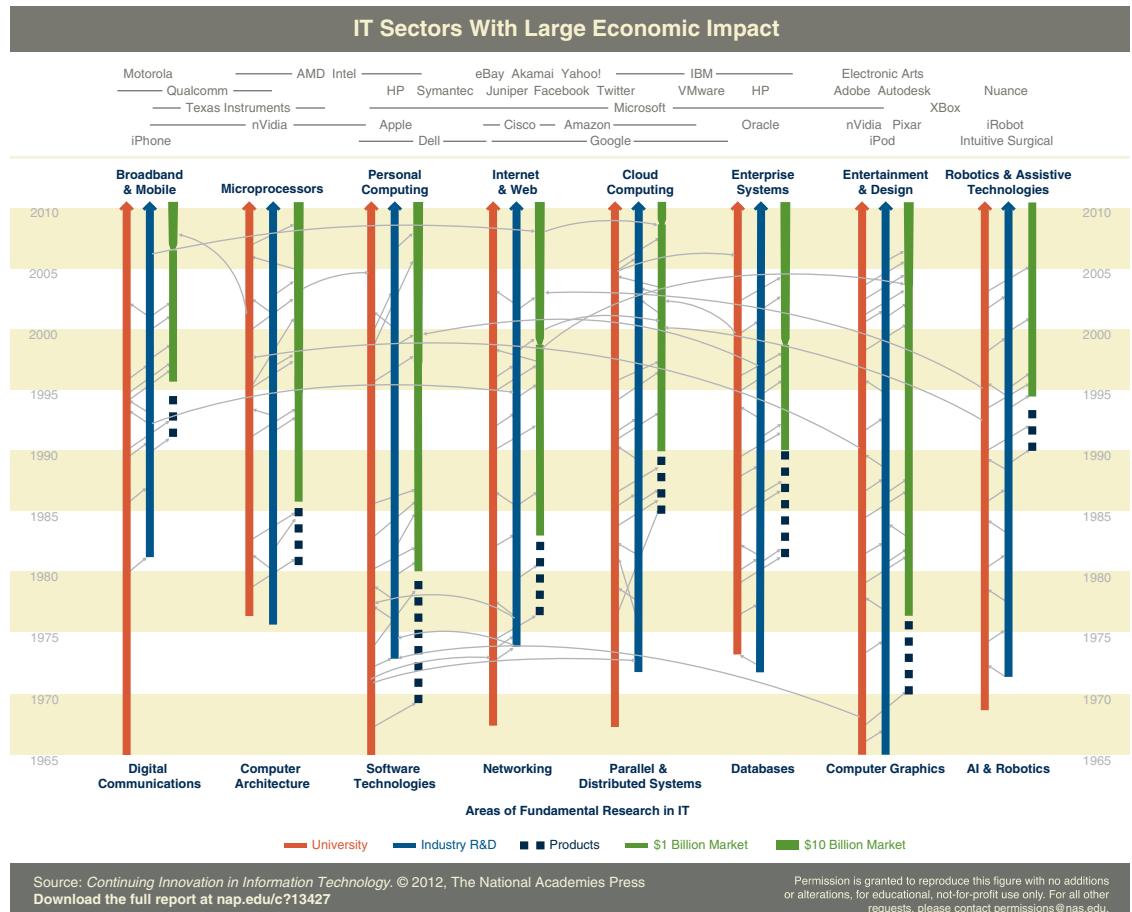


Fig. A.2. Tire Tracks diagram.

How to read this book

1. To appreciate the fundamentals of computer science

Read [Chapters 1, 2, 3, 4, 5](#), and [6](#), up to the Key Concepts summary in each chapter. These chapters take the reader from the beginnings of digital computers to a description of how computer hardware and software work together to solve a problem. The ideas of programming languages and software engineering are covered in [Chapter 4](#), and computer algorithms in [Chapter 5](#). [Chapter 6](#) is probably the most difficult chapter in the book as it tries to explain the fundamental theoretical insights of Alan Turing and Alonzo Church on computability and universality. This chapter can be skipped on a first reading without jeopardizing the understandability of the later chapters.

2. To learn about more of the early history of computing

Read all of the preceding, plus the history sections after the Key Concepts summaries in [Chapters 1](#) and [2](#). In [Chapter 1](#) we describe the very early ideas of Charles Babbage and Ada Lovelace, as well as the little-known Colossus computer, developed at the UK Post Office's Dollis Hill research laboratory, and LEO, the first business computer. The pioneering, independent computer developments of Konrad Zuse in Germany, Sergei Lebedev in Russia, and Trevor Pearcey in Australia are also summarized. [Chapter 2](#) gives more details about the first stored program computers, the Manchester Baby and the Cambridge EDSAC, developed in the United Kingdom as well as some history of computer memory technologies. In [Chapter 8](#) on the origins of personal computers there is also a history section describing the pioneers of interactive and personal computing as well as some insights on developments in computer architecture and some interesting anecdotes.

3. To understand Moore's law and semiconductor technologies

[Chapter 7](#) contains an account of the discovery of the transistor and the integrated circuit or silicon chip and the origins of Moore's law and Dennard

scaling. [Chapter 7](#) also contains a brief summary of the quantum mechanics of semiconductors. A fuller, but still elementary, account of quantum theory can be found in *The New Quantum Universe*, also published by Cambridge. [Chapter 15](#) looks at some future alternatives to silicon as the miniaturization level approaches atomic dimensions.

4. To understand the origins of personal computers, smart phones, and computer games

[Chapter 8](#) describes the development of personal computers based around microprocessors. This chapter looks at the origins of the WIMP environment and WYSIWYG word processors at Xerox Corporation's Palo Alto Research Center (PARC). It also describes the key roles played by IBM, Microsoft, and Apple in developing personal computers and briefly looks at the present era of smart phones, tablets, and touch interfaces. [Chapter 9](#) describes the origins of computer games and computer graphics.

5. To learn about the Internet, the World Wide Web, and search engines as well as the dangers of computer malware and hackers

The three key chapters are [Chapters 10, 11, and 12](#). [Chapter 10](#) describes the origin of the Internet with the ARPANET and packet switching. [Chapter 11](#) looks at the World Wide Web and hypertext and web browsers. It also includes an account of the PageRank algorithm and the rise in importance of Internet search engines and the social web. [Chapter 12](#) describes the history of computer malware with viruses, worms, and botnets. It also includes an introduction to cryptography, key exchange, and one-way functions.

6. To learn about the ideas of artificial intelligence (AI) and artificial neural networks and modern applications of machine learning applied to computer vision and natural language processing

These topics are covered in [Chapters 13 and 14](#) with a look to the future in [Chapter 16](#). [Chapter 13](#) describes early ideas about AI and the famous Turing Test. There is also an account of computer chess and IBM's Deep Blue machine and a summary of developments in artificial neural networks. [Chapter 14](#) starts with an introduction to Bayesian statistics before describing modern applications of machine learning technologies applied to computer vision, speech, and language processing. The chapter ends with a summary of IBM's Watson machine on the TV game show *Jeopardy!* The last chapter in this book looks to the future with an account of progress in robotics and the coming Internet of Things. [Chapter 16](#) ends with a look at "strong AI" and the problem of consciousness.

Notes

1. Beginnings of a revolution

1. Richard Feynman. *The Feynman Lectures on Computation*, edited by Tony Hey and Robin W. Allen. Perseus Books, 2000, xiii.
2. Herman Goldstine. *The Computer from Pascal to von Neumann*. Princeton University Press, 1972, 182.
3. *Ibid.*
4. Stan Augarten. *Bit by Bit: An Illustrated History of Computers*. Ticknor & Fields, 1984, 131, quotation attributed to Norris Bradbury.
5. John Mauchly. "Amending the ENIAC Story." *Datamation* 25, no. 11 (1979): 217.
6. Maurice Wilkes. *Memoirs of a Computer Pioneer*. MIT Press, 1985, 108.
7. B. E. Carpenter, Alan Turing, and Michael Woodger. *A.M. Turing's ACE Report of 1946 and Other Papers*. MIT Press, 1986, 21.
8. Martin Campbell-Kelly and William Aspray. *Computer: A History of the Information Machine*. Basic Books, 1996, 102.
9. Charles Babbage. *Passages from the Life of a Philosopher*. Longman, Green, Longman, Roberts, & Green, 1864, 42.
10. Philip Morrison and Emily Morrison (eds.). *Charles Babbage and His Calculating Engines*. Dover Publications, 1961, xiv.
11. Doron Swade. *The Difference Engine: Charles Babbage and the Quest to Build the First Computer*. Penguin Books, 2002, x.
12. Bernard Cohen. *Howard Aiken: Portrait of a Computer Pioneer*. MIT Press, 2000, 64.
13. Swade, *The Difference Engine*, 155.
14. Dorothy Stein. *Ada: Her Life and Her Legacy*. MIT Press, 1987, 82.
15. Luigi Menabrea. *Sketch of the Analytical Engine Invented by Charles Babbage*, printed by Richard and John E. Taylor, 1843, 696.
16. *Ibid.*, 713.
17. Ronald Lewin. *Ultra Goes to War, the Secret Story*. Arrow Books, 1980, 64.
18. Jack Good, Donald Michie, and Geoffrey Timms. *General Report on Tunny*. Public Record Office, 1945, 51.
19. BBC Radio 4. *Electronic Brains*, Programme 3 "Then We Took the Roof off," <http://www.bbc.co.uk/radio4/science/electronicbrains.shtml>.

20. Mike Hally. *Electronic Brains*. Granta Books, 2005, 168.

- B1. George Polya and G. L Alexanderson. *The Polya Picture Album: Encounters of a Mathematician*. Birkhäuser, 1987, 154.
- B2. William Poundstone. *Prisoner's Dilemma: John Von Neumann, Game Theory and the Puzzle of the Bomb*. Doubleday, 1992, 25.
- B3. Thomas Flowers. "The Design of Colossus," in Foreword by Howard Campaigne. *Annals of the History of Computing* 5 no. 3 (July 1983): 239.

2. The hardware

1. John Horgan. "Claude E. Shannon." *IEEE Spectrum* 29, no. 4 (April 1992): 72.
2. Daniel Hillis. *The Pattern on the Stone: The Simple Ideas That Make Computers Work*. Basic Books, 1999, 18.
3. *Ibid.*, 18.
4. Anthony Liversidge. "Claude E. Shannon, Interview: Father of the Electronic Information Age." *Omni Magazine* 9, no. 11 (August 1987): 65.
5. Robert Slater. *Portraits in Silicon*. MIT Press, 1989, 34, quotation from Claude Shannon.
6. Claude Shannon. "A Symbolic Analysis of Relay and Switching Circuits." *Transactions American Institute of Electrical Engineers* 57 (1938): 492.
7. Stan Augarten. *Bit by Bit: An Illustrated History of Computers*. Ticknor & Fields, 1984, 101.
8. Alan Perlis. "Epigrams on Programming," *ACM SIGPLAN Notices*, 17 (9) (September 1982): 9.
9. Mike Hally. *Electronic Brains: Stories from the Dawn of the Computer Age*. Granta Publications, 2005, 96, quotation from Freddie Williams.
10. Gerard O'Regan. *A Brief History of Computing*. Springer, 2008, 49.
11. Hally. *Electronic Brains: Stories from the Dawn of the Computer Age*. 96, quotation from Maurice Wilkes.
12. *Ibid.*, 94.
13. *Ibid.*, 97.
14. *Ibid.*, 90.
15. *Ibid.*
- B1. Clifford A. Pickover. *The Math Book*. Sterling, 2012, 242, quotation from Augustus de Morgan.

- B2. Stephen Graubard and Paul O. Leclerc, eds. *Books, Bricks, and Bytes: Libraries in the Twenty-First Century*. Transaction Publishers, 1997, 126.

3. The software is in the holes

1. Dave Andrews. "Knuth Comments on Code." *BYTE* 21, no. 9 (September 1996): 40.
2. Butler Lampson. "The Ongoing Computer Revolution." *The Bridge* 34, no. 2 (Summer 2004): 39.
3. John Dooley. *Software Development and Professional Practice*. APRESS ACADEMIC, 2011, 181, quotation from Maurice Wilkes.
4. Edsger Dijkstra. "EWD 1308: What Led to 'notes on structured programming,'" in *SoftwarePioneers*, edited by Manfred Broy and Ernst Denert. Springer-Verlag, 2002, 342.
5. Rob Kitchin and Martin Dodge. *Code/Space: Software and Everyday Life*. MIT Press, 2011, 3, quotation from Bjarne Stroustrup.
6. Keith Curtis. *After the Software Wars*. keithcu press, 2009, 209, quotation from Richard Feynman.
7. Paul Ceruzzi. *A History of Modern Computing*. MIT Press, 1998, 82. Original source given as Grace Hopper in "Computers and Their Future: Speeches Given at the World Computer Pioneers Conference," Llandudno, Wales, 1970.
8. Neil Barrett. *The Binary Revolution*. Weidenfeld and Nicholson, 2006, 211.
9. Martin Campbell-Kelly and William Aspray, *Computer: A History of the Information Machine*. Westview Press, 2004, 169.
10. John Backus. "The History of FORTRAN I, II, AND III," *ACM SIGPLAN Notices* 13, no. 8 (August 1978): 166.
11. *Ibid.*, 168.
12. Campbell-Kelly and Aspray, *Computer*, 171.
13. Michael Waldrop. *The Dream Machine: J.C.R. Licklider and the Revolution That Made Computing Personal*. Penguin Books, 2002, 164.
14. *Ibid.*, 174.
15. Ceruzzi, *A History of Modern Computing*, 148, quotation from Maurice Wilkes.

4. Programming languages and software engineering

1. Edsger Dijkstra. "The Humble Programmer." *Communications of the ACM* 15, no. 10 (1972): 861.
2. C. A. R. Hoare. "The Emperor's Old Clothes." *Communications of the ACM* 24, no. 2 (February 1981): 78.
3. Frederick Brooks Jr. *The Mythical Man-Month*. Addison-Wesley, 1995, 47.
4. Eric Raymond. *The Cathedral & the Bazaar*. O'Reilly Media, 2001, 19.
5. *Specifications for the IBM Mathematical FORmula TRANslating System, FORTRAN*. PRELIMINARY REPORT, Programming Research Group, Applied Science Division, International Business Machines Corporation, 10 November 1954, 2. http://archive.computerhistory.org/resources/text/Knuth_Don_X4100/PDF_index/k-7-pdf/k-7-u2763-Backus-Prelim-FORTRAN.pdf.
6. Edsger Dijkstra. "Go To Statement Considered Harmful." *Communications of the ACM* 11, no. 3 (March 1968): 147.
7. *Specifications for the IBM Mathematical FORmula TRANslating System*, 2.
8. Brooks, *The Mythical Man-Month*, 25.
9. IEEE Standard Glossary of Software Engineering Terminology. EEE std 610.12-1990, 1990, 67.
10. David Parnas and Paul Clements. "A Rational Design Process: How and Why to Fake It." *Journal IEEE Transactions on Software Engineering* 12, no. 2 (February 1986): 251.
11. David Harel. "Statecharts in the Making: A Personal Account." *Communications of the ACM* 52, no. 3 (2009): 70.
12. *MSDN Magazine*, February 2002, quotation from Grady Booch, <http://msdn.microsoft.com/en-us/magazine/cc301920.aspx>.
13. Edsger Dijkstra, "The Humble Programmer," 864.
14. NIST Planning Report 02-3-Summary - The Economic Impacts of Inadequate Infrastructure for Software Testing, NIST, 2002, 1, <http://samate.nist.gov/docs/econImpact-Summ.v23.pdf>.
15. Michael Cusumano and Richard Selby. *Microsoft Secrets*. 1st Touchstone ed. Simon and Schuster, 1988, 15.
16. Steve Ballmer. *Connecting with Customers*. 2 October 2002, <http://www.microsoft.com/mscorp/execmail/2002/10-02customers.mspx>.
17. Steve McConnell. *Code Complete*. 2nd ed. Microsoft Press, 2009, 521.
18. Empirical Software Engineering. <http://simula.no/research/se/emse>.
19. Steven Weber. *The Success of Open Source*. Harvard University Press, 2005, 28.
20. *Ibid.*, 47.
21. Linus Torvalds. "Free Minix-like Kernel Sources for 386-AT." Posted to newsgroup comp.os.minix, 5 October 1991. <https://groups.google.com/forum/#!msg/comp.os.minix/4995SivOl9o/GwqLJIPSICJ>.
22. Weber, *The Success of Open Source*, 69.
23. Quotation from the SourceForge website. <http://sourceforge.net/apps/trac/sourceforge/wiki/What%20is%20SourceForge.net>.
24. *Ibid.*
25. David Barron. *The World of Scripting Languages*. 1st ed. Wiley, 2000, 17.
26. Dennis Ritchie. *The Evolution of the Unix Time-Sharing System*. Bell Laboratories. <http://cm.bell-labs.com/who/dmr/hist.html>.
27. Brian Kernighan and Rob Pike. *The Unix Programming Environment*. Prentice Hall, 1984, viii.
28. Bill Gates. Keynote address. Windows Hardware Engineering Conference, 2002.
29. Edgar Codd. *The Relational Model for Database Management*. Addison-Wesley, 1990, iii.
30. Tony Long. "July 22, 1962: Mariner 1 Done In by a Typo." *Wired*, 2009, quotation from Arthur C. Clarke, http://www.wired.com/2009/07/dayintech_0722/.
31. Hoare, "The Emperor's Old Clothes," 83.
- B1. Edsger Dijkstra. "How Do We Tell Truths That Might Hurt?" In *Selected Writings on Computing: A Personal Perspective*. Springer-Verlag, 1982, 130. <http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD498.html>.

- B2. *Ibid.*
- B3. *Ibid.*
- B4. Joshua Gay, Richard Stallman, and Lawrence Lessig. *Free Software, Free Society: Selected Essays of Richard M. Stallman*. Createspace, 2009, 15.
- 5. Algorithms**
1. Charles Babbage. "Of the Analytical Engine," in *Passages from the Life of a Philosopher*, 1864.
 2. David Harel and Yishai Feldman. *Algorithmics: The Spirit of Computing*. 3rd ed. Addison Wesley, 2004, xii.
 3. Roger Eckhard. "Stan Ulam, John von Neumann, and the MONTE CARLO METHOD." *Los Alamos Science Special Issue* (1987): 131, quotation from Stanislaw Ulam, <http://library.lanl.gov/cgi-bin/getfile?15-13.pdf>.
 4. Thomas Misa. "An Interview with Edsger W. Dijkstra." *Communications of the ACM* 53 no.8 (2010): 42.
 5. "Drummer's Delight: The Shortest Way Around." *Newsweek*, 26 July 1954, 74.
 6. William Cook. *In Pursuit of the Traveling Salesman*. Princeton University Press, 2012, 94.
- B1. D. Shasha and C. Lazere. *Out of Their Minds: The Lives and Discoveries of 15 Great Computer Scientists*. Copernicus, 1998, 101, quotation from Donald Knuth.
- B2. John Francis. *Philosophy of Mathematics*. Global Vision Publishing House, 2008, 49, quotation from Richard Feynman.

6. Mr. Turing's amazing machines

1. Alan Turing. *A. Programmers' Handbook for the Manchester Electronic Computer Mark II*. 1952. <http://www.computer50.org/kgill/mark1/progman.html>.
 2. Andrew Hodges. *Alan Turing: The Enigma of Intelligence*. Vintage, 1992, 84.
 3. Scott Soames. *The Analytic Tradition in Philosophy*, Vol. 1, *The Founding Giants*. Princeton University Press, 2014, 124, quotation from Gottlob Frege's letter.
 4. Anne Rooney. *The History of Mathematics*. The Rosen Publishing Group, 2013, 201, quotation from David Hilbert.
 5. Hodges, *Alan Turing: The Enigma of Intelligence*, 91.
 6. Charles Petzold. *The Annotated Turing*. Wiley Publishing, 2008, 60.
 7. David Harel. *Computers Ltd: What They Really Can't Do*. Oxford University Press, 2000, 52.
 8. Oskar Morgenstern. "History of naturalization of Kurt Gödel." Institute for Advanced Study, 1971, 2. <http://cdm.itg.ias.edu/cdm/compoundobject/collection/coll12/id/2985/rec/3>.
 9. *Ibid*
 10. Hodges, *Alan Turing: The Enigma of Intelligence*, 145, quotation from Stanislaw Ulam.
 11. Jack Copeland. *The Essential Turing: The Ideas that Gave Birth to the Computer Age*. Oxford University Press, 2004, 22, quotation from Stanley Frankel.
 12. Petzold, *The Annotated Turing*, 167, quotation from John von Neumann.
 13. *Ibid.*, 163, quotation from Howard Aiken.
 14. Alan Turing. "Computing Machinery and Intelligence." *Mind* 59 (1950): 441.
 15. Hodges, *Alan Turing: The Enigma of Intelligence*, 113, quotation from Max Newman.
 16. *Ibid.*, 112, quotation from Max Newman.
 17. Petzold, *The Annotated Turing*, 63, quotation from Alonzo Church.
- B1. Rebecca Goldstein. *Incompleteness: The Proof and Paradox of Kurt Gödel*. W. W. Norton & Company, 2005, 223, quotation from John von Neumann.
- B2. *Ibid.*, 33, quotation from Oskar Morgenstern.
- B3. Dana Mackenzie. *The Universe in Zero Words: The Story of Mathematics as Told through Equations*. Princeton University Press, 2012, 189, quotation from Hilbert.

7. Moore's law and the silicon revolution

1. Tony Hey. *Feynman and Computation: Exploring the Limits of Computers*. Perseus Books Group, 1999, 23, quotation from Carver Mead.
2. Tony Hey and Patrick Walters. *The New Quantum Universe*. Cambridge University Press, 2003, 124, quotation from John Bardeen.
3. *Ibid.*, 125, quotation from Geoffrey Dummer.
4. Jack Kilby. "Invention of the Integrated Circuit." *IEEE Transactions on Electron Devices* 23, no. 7 (1976): 650.
5. Stan Augarten. *Bit by Bit*. Houghton Mifflin, 1984, 236, quotation from Jack Kilby.
6. Hey and Walters, *The New Quantum Universe*, 185, quotation from Robert Noyce.
7. Augarten, *Bit by Bit*, 243, quotation from Robert Noyce.
8. Ross Bassett. *To the Digital Age: Research Labs, Start-up Companies, and the Rise of MOS Technology*. Johns Hopkins University Press, 2002, 25.
9. Paul Ceruzzi. *A History of Modern Computing*. MIT Press, 2003, 187.
10. Gordon Moore. "Cramming More Components onto Integrated Circuits." *Electronics* 38, no. 8 (19 April 1965): 114.
11. Richard Rhodes. *Visions of Technology*. Touchstone, 1999, 224, quotation from Gordon Moore, SPIE 1995 speech, "Lithography and the Future of Moore's Law."
12. Hey and Walters, *The New Quantum Universe*, 187, quotation from Carver Mead.
13. V. K. Jain and Hemlata. *Enterprise Electronics & Mobile Commerce*. Cyber Tech Publications, 2001, 61, quotation from Arthur Rock.
14. Tekla Perry. "Morris Chang: Foundry Father." *IEEE Spectrum* 48, no. 5 (2011): 48, quotation from James Plummer.
15. *Ibid.*, 50, quotation from Jen-Hsung Huang.
16. Manek Dubash. "Moore's Law Is Dead, Says Gordon Moore." *Techworld* (13 April 2005), quotation from Gordon Moore, <http://news.techworld.com/operating-systems/3477/moores-law-is-dead-says-gordon-moore/>.
17. Hey and Walters, *The New Quantum Universe*, 107, quotation from Richard Feynman.
18. Quotation attributed to Seymour Cray.

19. Ibid.
20. Eric Raymond. *The New Hacker's Dictionary*. MIT Press, 1996, 270.
- B1. Gosta Ekspong. *Nobel Lectures, Physics: 1996–2000*. World Scientific Pub. Co. Inc., 2003, 472, quotation from Jack Kilby.
- B2. The United States Patent and Trademark Office. The National Medal of Technology and Innovation Recipients, 2002, <http://www.uspto.gov/about/nmti/recipients/2002.jsp>.

8. Computing gets personal

1. John Hennessy and David Patterson. *Computer Architecture: A Quantitative Approach*, Elsevier. 2012, 1, quotation from Bill Gates.
2. Mitchell Waldrop. *The Dream Machine: J. C. R. Licklider and the Revolution That Made Computing Personal*. Penguin Books, 2002, 145, quotation from Ken Olsen.
3. *Ibid.*, 146, quotation from Wesley Clark.
4. Michael Hiltzik. *Dealers of Lightning: Xerox PARC and the Dawn of the Computer Age*. HarperCollins Publishers, 1999, xxv.
5. Arthur Salsberg. “The Home Computer is here!” *Popular Electronics* (January 1975): 4.
6. Paul Ceruzzi. *A History of Modern Computing*. 2nd rev. ed. MIT Press, 2003, 228.
7. Paul Allen. *Idea Man*. Portfolio/Penguin, 2011, 9, quotation from Ed Roberts.
8. *Ibid.*
9. Ceruzzi, *A History of Modern Computing*, 234.
10. “The Home Computer That's Ready to Work, Play and Grow with You.” *Scientific American* 237, no. 3 (1977): 99.
11. Robert Slater. *Portraits in Silicon*. MIT Press, 1989, 285.
12. “The Birth of the IBM PC.” IBM archives. www-03.ibm.com/ibm/history/exhibits/pc25/pc25_birth.html.
13. Martin Campbell-Kelly and William Aspray. *Computer: A History of the Information Machine*. 2nd ed. Westview Press, 2004, 228.
14. Edward Bride. “The IBM Personal Computer: A Software-Driven Market.” *Computer* 44, no. 8 (2011): 34, quotation from Burton Grad.
15. *Ibid.*
16. David Bradley. “A Personal History of the IBM PC.” *Computer* 44, no. 8 (2011): 24.
17. *Ibid.*
18. *Ibid.*
19. Greg Goth. “IBM PC Retrospective: There Was Enough Right to Make It Work.” *Computer* 44, no. 8 (2011): 30, quotation from Mark Dean.
20. Douglas Smith and Robert Alexander, “Fumbling the Future: How Xerox Invented, Then Ignored, the First Personal Computer.” iUniverse.com (1999): 241, quotation from Steve Jobs.
21. Campbell-Kelly and Aspray, *Computer*, 241, quotation from John Sculley.
22. Walter Isaacson. *Steve Jobs*. Simon&Schuster Paperbacks, 2011, 129, quotation from Terry Oyama.
23. James Wallace and Jim Erickson. *Hard Drive: Bill Gates and the Making of the Microsoft Empire*. John Wiley, 1992, 252, quotation from Rowland Hanson.
24. Campbell-Kelly and Aspray, *Computer*, 246, quotation from Bill Gates.
25. *Ibid.*, 279.
26. *Computer Law Reporter* 20, nos. 1–3 (1994): 271.
27. Walter Isaacson. *Steve Jobs: The Exclusive Biography*. Little, Brown, 2013, 178, quotation from Bill Gates.
28. Hiltzik, *Dealers of Lightning*, 12.
29. Robert W. Taylor. “In Memoriam: J. C. R. Licklider 1915–1990.” Digital Research Center Research Report #61, Palo Alto, CA, 7 August 1990. <http://memex.org/licklider.pdf>.
30. Vannevar Bush. “As We May Think.” *Atlantic Monthly*, July 1945: 102.
31. *Ibid.*, 107.
32. Campbell-Kelly and Aspray, *Computer*, 237, quotation from Doug Engelbart.
33. Hiltzik, *Dealers of Lightning*, 17.
34. *Ibid.*, 382.
35. Stan Augarten. *Bit by Bit*. Houghton Mifflin, 1984, 257.
36. Butler Lampson, personal communication, 2013.
37. Ken Olsen, see <http://www.snopes.com/quotes/kenolsen.asp>.
38. Gordon Bell, personal communication, 2013.
39. Gordon Bell. In Edgar H. Schein, *DEC Is Dead, Long Live DEC: The Lasting Legacy of Digital Equipment Corporation*. Berrett-Koehler Publishers Inc., 2003, 2.
40. Hiltzik, *Dealers of Lightning*, xxii.
41. *Ibid.*, 168.
42. *Ibid.*, 200, quotation from Charles Simonyi.
43. *Ibid.*, 205, quotation from Tim Mott.
44. *Ibid.*, 228, quotation from Tim Mott.
45. *Ibid.*, 7, quotation from Butler Lampson.
46. *Ibid.*, 4, quotation from Chuck Thacker.
47. Chris Bidmead. “ARM Creators Sophie Wilson and Steve Furber, Part Two: The accidental chip.” *The Register*, May 2012, quotation from Sophie Wilson. http://www.theregister.co.uk/Print/2012/05/03/unsung_heroes_of_tech_arm_creators_sophie_wilson_and_steve_furber/.
48. Jason Fitzpatrick. “An Interview with Steve Furber.” *Communications of the ACM* 54, no. 5 (2011): 39.
49. David Bradley email from 2008, <http://mail.computerhistory.org/pipermail/inforoots/2008-April/001914.html>.
50. Wallace and Erickson, *Hard Drive*, 267.
- B1. Dan Bricklin. <http://www.bricklin.com/history/saiidea.htm>.
- B2. Plaque in Aldrich 108 at Harvard Business School. <http://www.bricklin.com/history/saiiproduct1.htm>.
- F1. Robert Metcalfe and David Boggs. “Ethernet: Distributed Packet Switching for Local Computer Networks.” *Communications of the ACM* 19, no. 7 (1976): 395.
- F2. Paul Freiberger. “History of Microcomputing, Part 1: Homebrew Club.” *InfoWorld* 4, no. 7 (1982): 17.
- F3. Computer History Museum website. <http://www.computerhistory.org/revolution/personal-computers/17300>.
- F4. ACM A. M. Turing Award website. <http://awards.acm.org/amturing/all.cfm>.

- F5. John Markoff. "A Maverick Scientist Gets an I.B.M. Tribute." *The New York Times*, 26 June 1990.
- F6. Ute Bradley. *Applied Marketing and Social Research*. J. Wiley, 1987, 114.
- F7. ACM A. M. Turing Award website. <http://awards.acm.org/amturing/all.cfm>.

9. Computer games

1. David Evans, Andrei Hagiuc, and Richard Schmalensee. *Invisible Engines*. MIT Press 2006, 115, quotation from Shigeru Miyamoto.
2. Graham Nelson. *Inform Designer's Manual*. 4th ed. The Interactive Fiction Library, 2001, 343, quotation from William Crowther.
3. Mark Wolf. *The Video Game Explosion*. Greenwod Publishing, 2008, 71.
4. Chris Kohler. *Power-Up: How Japanese Video Games Gave the World an Extra Life*. BradyGAMES/Pearson Education, 2005, 56.
5. David Sheff. *Game Over: How Nintendo Conquered the World*. Vintage Books, 1994, 51.
6. Allison Cerra and Christina James. *Identity Shift: Where Identity Meets Technology in the Networked-Community Age*. John Wiley & Sons, 2011, 168.
7. Computer Gaming World's Anniversary Edition, 1996. <http://www.cdaccess.com/html/pc/150best.htm>.
8. Tom Cheshire. "Want to Learn Computer-Aided Design (CAD)? Play Minecraft." *Wired Magazine* (UK), November 2012, quotation from Cody Sumter. <http://www.wired.co.uk/magazine/archive/2012/11/play/minecrafted>.
9. "Google Quantum AI. Lab Team." 18 October 2013. <https://plus.google.com/+QuantumAILab/posts/grMbaaDGChH>.
10. Guinness World Records 2014: Gamer's Edition. Guinness World Records Ltd., 2014.
11. Tom Cheshire. "In Depth: How Rovio Made Angry Birds a Winner (and What's Next)." *Wired*, 7 March 2011, quotation from Niklas Head. <http://www.wired.co.uk/magazine/archive/2011/04/features/how-rovio-made-angry-birds-a-winner>.
- F1. *Ibid.*, quotation from Niklas Head.
- B1. Wolf, *The Video Game Explosion*, 71.
- B2. Tom Hoggins. "Interview: Meeting Shigeru Miyamoto." *Telegraph*, 28 October 2008, http://blogs.telegraph.co.uk/technology/tomhoggins/5525247/Interview_Meeting_Shigeru_Miyamoto/.

10. Licklider's Intergalactic Computer Network

1. J. C. R. Licklider. "Man-Computer Symbiosis." *IRE Transactions on Human Factors in Electronics*, HFE-1 (1960): 4. <http://memex.org/licklider.pdf>.
2. Stephen Segaller. NERDS 2.0.1. TV Books, 1998, 40, quotation from Larry Roberts.
3. Segaller, NERDS 2.0.1, 235, quotation from Scott McNealy.
4. Tom Standage. *The Victorian Internet: The Remarkable Story of the Telegraph and the Nineteenth Century's On-line Pioneers*. Phoenix, 1999, 2.
5. Stewart Brand. "Founding Father." *Wired* 9, no. 3 (2001), interview with Paul Baran. <http://www.wired.com/wired/archive/9.03/baran.html>.
6. Katie Hafner and Matthew Lyon. *Where Wizards Stay Up Late: The Origins of the Internet*. Touchstone, 1998, 55.
7. Brand, "Founding Father."
8. Hafner and Lyon, *Where Wizards Stay Up Late*, 59.
9. Brand, "Founding Father."
10. *Ibid.*
11. Donald Davies. "An Historical Study of the Beginnings of Packet Switching." *The Computer Journal* 44 (2001): 160.
12. Janet Abbate. "From Arpanet to Internet: A History of ARPA-Sponsored Computer Networks, 1966–1988." PhD dissertation, University of Pennsylvania, 1994, 41, quotation from Larry Roberts.
13. Davies, "An Historical Study of the Beginnings of Packet Switching," 157.
14. Hafner and Lyon, *Where Wizards Stay Up Late*, 38.
15. Campbell-Kelly and Aspray, *Computer*, 261.
16. Segaller, NERDS 2.0.1, 60.
17. Hafner and Lyon, *Where Wizards Stay Up Late*, 80.
18. *Ibid.*, 86, quotation from Leo Beranek.
19. *Ibid.*, 84, quotation from Leo Beranek.
20. BBN website. <http://www.bbn.com/resources/pdf/arpanet04.01.05.pdf>.
21. Hafner and Lyon, *Where Wizards Stay Up Late*, 122, quotation from Will Crowther.
22. *Ibid.*, 123, quotation from Will Crowther.
23. Segaller, NERDS 2.0.1, 89, quotation from Vint Cerf.
24. Hafner and Lyon, *Where Wizards Stay Up Late*, 146, quotation from Vint Cerf.
25. *Ibid.*, 130, quotation from Bob Kahn.
26. *Ibid.*, 157, quotation from Bob Kahn.
27. Segaller, NERDS 2.0.1, 95, quotation from Larry Roberts.
28. *Ibid.*, 104, quotation from Ray Tomlinson.
29. *Ibid.*, 105, quotation from Bob Kahn.
30. *Ibid.*, 106, quotation from Ray Tomlinson.
31. Hafner and Lyon, *Where Wizards Stay Up Late*, 238.
32. *Ibid.*, 237, quotation from Jon Postel.
33. Alec Gambling. Invited talk at 50th Anniversary Symposium for the Department of Electronics at the University of Southampton, 1997.
34. *Laser Focus World*, Vol. 2. International Data Publishing Company, April 1966, 144, www.sff.net/people/jeff.hecht/history.html.
35. Jeff Hecht. *City of Light: The Story of Fiber Optics*. Oxford University Press, 1999, 117, quotation from Charles Kao.
36. David Payne. Invited talk at 50th Anniversary Symposium for the Department of Electronics at the University of Southampton, 1997.
37. Anonymous. "The Central Telegraph Office." *Illustrated London News*, November 1874, 506.
- B1. The Nobel Prize in Physics 2009. Press Release 6 October 2009, http://www.nobelprize.org/nobel_prizes/physics/laureates/2009/press.html.
- B2. The Franklin Institute Awards. 1998, <http://archive.today/ixBRZ>.

- F1. Taylor, “In Memoriam: J. C. R. Licklider 1915–1990,” 34, <http://memex.org/licklider.pdf>.
 F2. *Ibid.*, 26.

11. Weaving the World Wide Web

1. Tim Berners-Lee. Proposal for the World Wide Web, 1989. www.w3.org/History/1989/proposal.html.
2. Bush, “As We May Think,” 101.
3. *Ibid.*
4. *Ibid.*, 106.
5. *Ibid.*, 107.
6. Ted Nelson. “Ted Nelson’s Computer Paradigm Expressed as One-Liners.” 1999. <http://xanadu.com.au/ted/TN/WRITINGS/TCOMPARADIGM/tedCompOneLiners.html>.
7. Bush, “As We May Think,” 108.
8. Jean Armour-Polly. “Surfing the Internet.” *University of Minnesota Wilson Library Bulletin*, June 1992.
9. Tim Berners-Lee. *Weaving the Web*. Orion Business Books, 1999, 11.
10. *Ibid.*, 16.
11. *Ibid.*, 26.
12. James Gillies and Robert Cailliau. *How the Web Was Born: The Story of the World Wide Web*. Oxford Paperbacks, 2000, 196.
13. Berners-Lee, *Weaving the Web*, 30.
14. *Ibid.*, 56.
15. *Ibid.*, 80.
16. Bill Gates. “Internet Tidal Wave,” memo. May 1995, <http://www.lettersofnote.com/2011/07/internet-tidal-wave.html>.
17. *Ibid.*
18. Mozilla Foundation website. <https://www.mozilla.org/en-US/foundation/>.
19. David Vise. *The Google Story*. Delacourt Press, 2008, 37.
20. Harry Shum. January 2011. http://www.bing.com/community/site_blogs/b/search/archive/2011/02/01/thoughts-on-search-quality.aspx.
21. Darcy DiNucci. “Fragmented Future.” *Print* 53, no. 4 (1999): 32. http://darcydi.com/fragmented_future.pdf.
22. Ward Cunningham. “What Is Wiki?” 2002. <http://www.wiki.org>.
23. Lars Aronsson. Quoted in Wikipedia article on Wiki. <http://en.wikipedia.org/wiki/Wiki>.
24. Tim Berners-Lee. “developerWorks Interviews: Tim Berners-Lee.” *developerWorks*, 22 August 2006, <http://www.ibm.com/developerworks/podcast/dwi/cm-int082206txt.html>.
25. Tim Berners-Lee, James Hendler, and Ora Lassilla. “The Semantic Web.” *Scientific American*, 17 May 2001, 30.
- B1. Pascal Zachary. *Endless Frontier: Vannevar Bush, Engineer of the American Century*. MIT Press, 1999, 125.

12. The dark side of the web

1. David Sanger. *Confront and Conceal: Obama’s Secret Wars and Surprising Use of American Power*. Broadway Books, 2013, xiv.
2. *The Capstone Encyclopaedia of Business*. Capstone Publishing, 2003, 444, definition by Fred Cohen.

3. Merrill Chapman. *In Search of Stupidity: Over Twenty Years of High Tech Marketing Disasters*. APRESS, 2006, 262, quotation from Thomas Hesse.
4. Bob Brown. “Sony BMG Rootkit Scandal: 5 Years Later.” *NetworkWorld*, 1 November 2010, quotation from Mikko Hypponen, <http://www.networkworld.com/article/2194292/network-security/sony-bmg-rootkit-scandal-5-years-later.html>.
5. Clifford Stoll. *The Cuckoo’s Egg*. Pan Books, 1991, 371.
6. “Microsoft Security Intelligence Report Warns of Ongoing Conficker Threat and Clarifies Reality of Targeted Attacks.” 25 April 2012, <http://www.microsoft.com/en-us/news/press/2012/apr12/04-25sirvapr12pr.aspx>.
7. Richard Boscovich. “Microsoft Works with Financial Services Industry Leaders, Law Enforcement and Others to Disrupt Massive Financial Cybercrime Ring.” 5 June 2013, http://blogs.technet.com/b/microsoft_blog/archive/2013/06/05/microsoft-works-with-financial-services-industry-leaders-law-enforcement-and-others-to-disrupt-massive-financial-cybercrime-ring.aspx.
8. The Official Microsoft Blog. “Microsoft Disrupts the Emerging Nitol Botnet Being Spread through an Unsecure Supply Chain.” 13 September 2012, http://blogs.technet.com/b/microsoft_blog/archive/2012/09/13/microsoft-disrupts-the-emerging-nitol-botnet-being-spread-through-an-unsecure-supply-chain.aspx.
9. Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography: Principles and Protocols*. Chapman & Hall/CRC, 2008, 10.
10. Simon Singh. *The Code Book*. Fourth Estate Limited, 1999, 267.
11. *Ibid.*, 252.
12. *Ibid.*, 266.
13. Text adapted from Singh, *The Code Book*, 266.
14. *Ibid.*, 308, quotation from Ron Rivest.
15. Monty Python. “Spam,” Episode 25, December 1970, <http://www.montypython.net/scripts/spam.php>.
16. Singh, *The Code Book*, 256.
17. *Ibid.*, 288.

13. Artificial intelligence and neural networks

1. Herbert Simon and Allen Newell. “Heuristic Problem Solving: The Next Advance in Operations Research.” *Operations Research* 6, no. 1, INFORMS 1958, 8.
2. Norbert Wiener. *Cybernetics, or Control and Communication in the Animal and the Machine*. 2nd ed. MIT Press, 1961, 43.
3. Alan Turing. “Computing Machinery and Intelligence.” Reprinted in *The Essential Turing*, edited by B. Jack Copeland. Clarendon Press, 2005, 441.
4. Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd ed. Prentice Hall, 2010, 2.
5. Turing, “Computing Machinery and Intelligence,” 449.
6. *Ibid.*, 395.
7. *Ibid.*, 452.
8. John McCarthy et al. “A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence.” 31 August 1955.

9. Allen Newell, J. Shaw, and Herbert Simon. "Elements of a Theory of Human Problem Solving." *Psychological Review* 65, no. 3 (1958): 156.
10. Allen Newell, J. Shaw, and Herbert Simon. "The Processes of Creative Thinking." *RAND Corporation Report P-1320*, 1958.
11. John McCarthy. "Programs with Common Sense," in *Semantic Information Processing*. MIT Press, 1968, 403.
12. Russell and Norvig, *Artificial Intelligence: A Modern Approach*, 25, quotation from David McAllester.
13. Donald Michie. *On Machine Intelligence*. 2nd ed. Ellis Horwood Ltd., 1986, 78.
14. David Levy. "Computer Chess – Past, Present and Future." *Chess Life & Review*, December 1973, 726.
15. David Levy. *More Chess and Computers: The Microcomputer Revolution, the Challenge Match*. Computer Science Press, 1980, 30.
16. Garry Kasparov. "IBM Owes Me a Rematch." *Time* 149, no. 21 (1997): 66.
17. Hippocrates. *On the Sacred Disease*, translated by Francis Adams. <http://classics.mit.edu/Hippocrates/sacred.html>.
18. Mo Costandi. "The Discovery of the Neuron." 2006, quotation from Ramón y Cajal, <http://neurophilosophy.wordpress.com/2006/08/29/the-discovery-of-the-neuron/>.
19. Lisa Rezende. *Chronology of Science*. Checkmark Books, 2007, 244.
20. Charles Sherrington. *Man on His Nature*. Cambridge University Press, 1942, 178.
21. Turing, "Computing Machinery and Intelligence," 405.
22. Russell and Norvig, *Artificial Intelligence: A Modern Approach*, 761.
23. Larry Crockett. *The Turing Test and the Frame Problem*. Ablex Publishing Corporation, 1994, 183.

14. Machine learning and natural language processing

1. Warren Weaver's letter to Norbert Wiener. 1949, 4, <http://www.mt-archive.info/Weaver-1949.pdf>.
2. Jamie MacLennan, ZhaoHui Tang, and Bogdan Crivat. *Data Mining with SQL Server 2008*. Wiley Publishing, 2009, 216.
3. Sharon McGayne. *The Theory That Would Not Die*. Yale University Press, 2011, 25.
4. Irving Good. *Good Thinking: The Foundations of Probability and Its Applications*. Dover Publications, 2009, 124.
5. McGayne, *The Theory That Would Not Die*, 220, quotation from Adrian Smith.
6. McGayne, *The Theory That Would Not Die*, 242, quotation from David Heckerman.
7. "Alumnus Dr Jamie Shotton and the Development of Kinect for Xbox 360." Cambridge University Engineering Department. 20 January 2011, news quotation from Jamie Shotton, http://www.eng.cam.ac.uk/news/stories/2011/Xbox_Kinect/.
8. *Ibid.*
9. Stephen Baker. *Final Jeopardy*. Houghton Mifflin Harcourt, 2011, 7, quotation from Paul Horn.
10. Cynthia Lowry. "Merv Griffin: Question and Answer Man," *Independent Star-News*, Associated Press, 29 March 1964.
11. Baker, *Final Jeopardy*, 123, quotation from David Ferrucci.

12. *Ibid.*, 130, quotation from David Ferrucci.
13. *Ibid.*, 251, quotation from Ken Jennings.
14. *Ibid.*, 14, quotation from David Ferrucci.
15. Robert Wilson and Frank Keil. *The MIT Encyclopedia of the Cognitive Sciences*. MIT Press, 2001, 115, quotation from John Searle.
16. John Searle. "Watson Doesn't Know It Won on Jeopardy!" *Wall Street Journal*, 23 February 2013, <http://online.wsj.com/article/SB10001424052748703407304576154313126987674.html>.

15. The end of Moore's law

1. Gordon Moore. "Cramming More Components onto Integrated Circuits." *Proceedings of the IEEE* 86, no. 1 (January 1998): 84.
2. Richard Feynman. "There's Plenty of Room at the Bottom." Transcript of a talk presented to the American Physical Society, in Pasadena, California, December 1959.
3. *Ibid.*
4. *Ibid.*
5. BBC News website. "IBM Researchers Make 12-Atom Magnetic Memory Bit." 13 January 2012, quotation from Sebastian Loth, <http://www.bbc.co.uk/news/technology-16543497>.
6. Manek Dubash. "Moore's Law Is Dead, Says Gordon Moore." *Techworld*, 13 April 2005, <http://news.techworld.com/operating-systems/3477/moores-law-is-dead-says-gordon-moore/>.
7. ITC. "International Technology Roadmap for Semiconductors 2013 Edition ITC Overview." 6, <http://www.public.itrs.net/Links/2013ITRS/2013Chapters/2013Overview.pdf>.
8. *Ibid.*, 14.
9. Alistair Kemp. "Intel Reinvents Transistors Using New 3-D Structure." Intel UK Newsroom, 4 May 2011. http://newsroom.intel.com/community/en_uk/blog/2011/05/04/intel-reinvents-transistors-using-new-3-d-structure.
10. Karen Rhodes. "Radical New Intel Transistor Based on UC Berkeley's FinFET." <http://vcresearch.berkeley.edu/news/radical-new-intel-transistor-based-uc-berkeley-s-finfet>.
11. Michio Kaku. "What Happens When Computers Stop Shrinking?" *Salon*, 19 March 2011, http://www.salon.com/2011/03/19/moores_law_Ends_Excerpt/.
12. The Nobel Prize in Physics 2010. [nobelprize.org](http://www.nobelprize.org/nobel_prizes/physics/laureates/2010/), http://www.nobelprize.org/nobel_prizes/physics/laureates/2010/.
13. Richard Feynman. "Simulating Physics with Computers." *International Journal of Theoretical Physics* 21, nos. 6–7 (1982): 486.
14. *Ibid.*, 474.
15. Ervin Schrödinger. "Discussion of Probability Relations between Separated Systems." *Proceedings of the Cambridge Philosophical Society* 31 (1935): 555.
16. Brian Clegg. *The God Effect: Quantum Entanglement, Science's Strangest Phenomenon*. St. Martin's Griffin, 2009, 3, quotation from Einstein's letter to Max Born, 3 March 1947.
17. Chris Monroe and Jungsang Kim. "Scaling the Ion Trap Quantum Processor." *Science* 339, no. 6124 (2013): 1165.

18. Tom Knight. "Taming Biology as an Engineering Technology." Yale Department of Electrical Engineering Seminar, 12 November 2009, <http://robotics.research.yale.edu/index.php?n>Main.2009>.
- B1. The Kavli Foundation website. <http://www.kavli.foundation.org/2010-nanoscience-citation>.
- 16. The third age of computing**
1. Butler Lampson. "What Computers Do: Model, Connect, Engage." *Theory and Applications of Models of Computation*. Lecture Notes in Computer Science 7287 (2012): 23.
 2. *Ibid.*, 24.
 3. *Ibid.*, 25.
 4. George Devol. "Programmed Article Transfer." U.S. Patent 2988237, 1961, <http://www.google.com/patents/US2988237>.
 5. Kevin Ashton. "That 'Internet of Things' Thing." *RFID Journal*, 22 July 2009. <http://www.rfidjournal.com/articles/view?4986>.
 6. Ray Ozzie. "Dawn of a New Day." Memo to Microsoft staff, 28 October 2010. <http://ozzie.net/docs/dawn-of-a-new-day/>.
 7. *Ibid.*
 8. Russell and Norvig, *Artificial Intelligence* 1020.
 9. McCarthy et al. "Proposal for the Dartmouth Summer Research Project on Artificial Intelligence," 1.
 10. Russell and Norvig, *Artificial Intelligence*, 30.
 11. Russell and Norvig, *Artificial Intelligence*, 1020.
 12. Francis Crick. *The Astonishing Hypothesis: The Scientific Search for the Soul*. Simon & Schuster, 1995, 271.
 13. Stevan Harnad. Personal communication. April 2013.
 14. *Ibid.*
 15. Alan Turing. "Computing Machinery and Intelligence." *Mind* 59, no. 236 (1950): 444.
 16. Jeff Hawkins and S. Blakeslee. *On Intelligence*. St. Martin's Press, 2004, 68.
 17. *Ibid.*, 69.
 18. *Ibid.*, 104.
 19. *Ibid.*, 105.
 20. David Dennett. *Consciousness Explained*. Penguin Books, 1993, 73.
 21. Chris Koch. *Consciousness: Confessions of a Romantic Reductionist*. MIT Press, 2012, 27.
 22. *Ibid.*, 33.
 23. Dennett, *Consciousness Explained*, 21.
 5. John Malone. *Predicting the Future: From Jules Verne to Bill Gates*. M. Evans & Co Inc., 1997, 3, quotation from Pierre-Jules Hetzel.
 6. Jules Verne. *Paris in the Twentieth Century*. Ballantine Books, 1996, 51.
 7. *Ibid.*, 53.
 8. Gary Westfahl. *Hugo Gernsback and the Century of Science Fiction*. McFarland & Co Inc., 2007, 20, quotation from Hugo Gernsback.
 9. John Campbell. "The Last Evolution," in *Classic Science Fiction*, edited by Lester Del Rey. Ballantine Books, 1976, 2.
 10. Isaac Asimov. *The Early Asimov*, Volume 1. Victor Gollancz Ltd., 1973, 14.
 11. David Seed. *Science Fiction: A Very Short Introduction*. Oxford University Press, 2011, 120, quotation from John Campbell.
 12. Kurt Vonnegut. "EPICAC." *Welcome to the Monkey House*. Granada Publishing, 1968, 257.
 13. Isaac Asimov. *Robot Dreams*. Ace Books, 1990, xv.
 14. Murray Leinster. "A Logic Named Called Joe." *Astounding Science Fiction*, 1946. Republished in Billee J. Stallings and Jo-an J. Evans, *Murray Leinster Life and Works*. McFarland, 2011, 176.
 15. Josh Lowensohn. "Samsung Cites Kubrick Film in Apple Patent Spat." C/Net, 23 August 2011. http://news.cnet.com/8301-27076_3-20096061-248/samsung-cites-kubrick-film-in-apple-patent-spat.
 16. Asimov, *The Early Asimov*, Volume 1, 150.
 17. Isaac Asimov. "Runaround," *Astounding Science Fiction*, March 1942.
 18. Michael Anderson and Susan Leigh Anderson. *Machine Ethics*. Cambridge University Press, 259, quotation from Isaac Asimov.
 19. Isaac Asimov. *Foundation and Earth*. Doubleday and Company, Inc, 1986, 347.
 20. Red Dwarf. Season IV, Episode 5, "Dimension Jump." http://www.cervenytrpaslik.cz/scenare/EN-23-4_Dimension_Jump.htm.
 21. Douglas Adams. *The Restaurant at the End of the Universe (Hitchhiker's Guide 2)*. Pan Books, 1999, 34.
 22. *Ibid.*, 44.
 23. Greg Bear. *Queen of Angels*. Warner Books, 1991, 38, quotation from Martin Burke.
 24. *Ibid.*
 25. Neal Stephenson. *The Diamond Age*. Reissue ed. Penguin, 2011, 64.
 26. *Ibid.*, 56.
 27. Michael Crichton. *Prey*. Avon Books, 2003, 201.
 28. Michael Crichton. *Timeline*. Arrow Books, 2000, 138.
 29. *Ibid.*, 445.
 30. William Gibson and Bruce Sterling. *The Difference Engine*. Gollancz, 2011, 126.
 31. *Ibid.*, 132.
 32. *Ibid.*, 172.
 33. *Ibid.*, 376.
 34. *Ibid.*, 377.
 35. John Carlin. "A Farewell to Arms," *Wired*, May 1997, <http://archive.wired.com/wired/archive/5.05/netizen.html>.

36. Mark Russinovich. *Zero Day*. Thomas Dunne Books, 2011, 327.
37. *Ibid.*, 93.
38. William Gibson. *Neuromancer*. Voyager paperback ed., HarperCollins Publishers, 1995, 67.
39. Marvin Minsky. <http://web.media.mit.edu/~minsky/papers/option.chapters.txt>.
40. Harry Harrison and Marvin Minsky. *The Turing Option*. Viking Penguin Group, 1992, 70.
41. *Ibid.*, 315.
42. *Ibid.*, 316.
43. Marvin Minsky. *The Society of Mind*. Simon & Schuster, 1988, 308.
44. Source: <http://en.wikipedia.org/wiki/Cortana>.
45. Greg Iles. *The Footprints of God*. Pocket Star Books, 2003, 78.
46. *Ibid.*, 81.
- B1. Richard Dawkins. *A Devil's Chaplain: Selected Writings*. Phoenix, 2004, 195.

Epilogue

1. Alan Turing. “Intelligent Machinery, A Heretical Theory.” Reprinted in *The Essential Turing*, edited by Jack Copeland, Clarendon Press, 2005, 474.
2. J. C. R. Licklider, “Man-Computer Symbiosis,” 4.
3. Blay Whitby. *Reflections on Artificial Intelligence*. Intellect Books, 1996, 20, quotation from Marvin Minsky.

Suggested reading

1. Overviews of computer science

The following books provide a broad overview of different aspects of computing at an accessible level:

Hal Abelson, Ken Ledeen, and Harry Lewis. *Blown to Bits: Your Life, Liberty, and Happiness After the Digital Explosion*. Pearson Education, Inc., 2008.

Stan Augarten. *Bit by Bit: An Illustrated History of Computers*. Ticknor & Fields, 1984.

Neil Barrett. *The Binary Revolution*. Weidenfeld & Nicolson, 2006.

Martin Campbell-Kelly and William Aspray. *Computer: A History of the Information Machine*. Westview Press, 2004.

David Harel. *Algorithmics: The Spirit of Computing*. Addison Wesley, 3rd ed., 2004.

Daniel Hillis. *The Pattern on the Stone: The Simple Ideas That Make Computers Work*. Basic Books, 1998.

2. More detailed reading about specific topics

Paul Allen. *Idea Man*. Portfolio/Penguin, 2011.

Gordon Bell and Jim Gemmell. *Total Recall: How the E-Memory Revolution Will Change Everything*. Penguin Group USA, 2009.

Tim Berners-Lee. *Weaving the Web*. Orion Business Books, 1999.

Paul E. Ceruzzi. *A History of Modern Computing*. MIT Press, 1998.

William J. Cook. *In Pursuit of the Traveling Salesman*. Princeton University Press, 2012.

George Dyson. *Turing's Cathedral: The Origins of the Digital Universe*. Pantheon Books, 2012.

Richard Feynman. *The Feynman Lectures on Computation*, edited by Tony Hey and Robin W. Allen. Perseus Books, 2000.

Katie Hafner and Mathew Lyon. *Where Wizards Stay Up Late: The Origins of the Internet*. Touchstone, 1998.

David Harel. *Computers Ltd: What They Really Can't Do*. Oxford University Press, 2000.

Michael Hiltzik. *Dealers of Lightning: Xerox PARC and the Dawn of the Computer Age*. HarperCollins Publishers, 1999.

Andrew Hodges. *Alan Turing: The Enigma of Intelligence*. Unwin Paperbacks, 1983.

Tracy Kidder. *The Soul of a New Machine*. Little, Brown and Company, 1981.

John MacCormick. *9 Algorithms that Changed the Future: The Ingenious Ideas that Drive Today's Computers*. Princeton University Press, 2012.

Sharon Bertsch McGrayne. *The Theory that Would Not Die*. Yale University Press, 2011.

Nate Silver. *The Signal and the Noise: Why So Many Predictions Fail – But Some Don't*. Penguin Group USA, 2012.

Clifford Stoll. *The Cuckoo's Egg*. Pan Books, 1991.

Doron Swade. *The Difference Engine: Charles Babbage and the Quest to Build the First Computer*. Penguin Books, 2002.

Robert Slater. *Portraits in Silicon*. MIT Press, 1987.

Mitchell Waldrop. *The Dream Machine: J. C. R. Licklider and the Revolution that Made Computing Personal*. Penguin Group USA, 2002.

James Wallace and Jim Erickson. *Hard Drive: Bill Gates and the Making of the Microsoft Empire*. John Wiley, 1992.

3. Further reading by chapter

Chapter 1

- James Essinger. *Jacquard's Web*. Oxford University Press, 2004.
- Mike Hally. *Electronic Brains: Stories from the Dawn of the Computer Age*. Granta Publications, 2005.
- F. H. Hinsley and Alan Stripp (eds.). *Codebreakers: The Inside Story of Bletchley Park*. Oxford University Press, 1993.
- Brenda Maddox. *A Computer Called Leo*. Harper Perennial, 2004.
- Simon Winchester. *The Map that Changed the World*. Viking, 2001.
- Konrad Zuse. *The Computer – My Life*. Springer-Verlag, 1993.

Chapter 2

- John Hennessy and David Patterson. *Computer Architecture: A Quantitative Approach*. Elsevier and Morgan Kaufmann Publishers, 4th edition, 2006.
- Warren Fenton Stubbins. *Essential Electronics*. John Wiley & Sons, 1986.

Chapter 3

- J. Glenn Brookshear. *Computer Science: An Overview*. Addison-Wesley, 11th edition, 2012.
- Maurice Wilkes. *Memoirs of a Computer Pioneer*. MIT Press, 1985.
- Thomas J. Watson Jr. *Father, Son & Co. – My Life at IBM and Beyond*. Bantam, 1990.

Chapter 4

- David Barron. *The World of Scripting Languages*. John Wiley & Sons, 2000.
- Fred Brooks. *The Mythical Man Month*. Addison-Wesley, 1982.
- Michael Cusmano and Richard Selby. *Microsoft Secrets*. Touchstone Edition, 1998.
- Steve McConnell. *Code Complete: A Practical Handbook of Software Construction*. Microsoft Press, 2004.
- Eric Raymond. *The Cathedral and the Bazaar*. O'Reilly Media, 1999.
- Ian Sommerville. *Software Engineering*. Addison-Wesley, 6th edition, 2001.

Steve Weber. *The Success of Open Source*. Harvard University Press, 2004.

Chapter 5

- Alfred Aho, John Hopcroft, and Jeffrey Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1987.
- Ira Pohl and Alan Shaw. *The Nature of Computation: An Introduction to Computer Science*. Computer Science Press, 1981.

Chapter 6

- Martin Davis. *The Universal Computer: The Road from Leibniz to Turing*. W. W. Norton & Company, 2000.
- B. Jack Copeland. *The Essential Turing: The Ideas that Gave Birth to the Computer Age*. Oxford University Press, 2004.
- Charles Petzold. *The Annotated Turing*. Wiley Publishing, 2008.

Chapter 7

- Michael Riordan and Lillian Hoddeson. *Crystal Fire: The Invention of the Transistor and the Birth of the Information Age*. W. W. Norton & Company, 1998.

Chapter 8

- Paul Freiberger and Michael Swaine. *Fire in the Valley: The Making of the Personal Computer*. McGraw-Hill Publishing, 2nd revised edition, 2000.
- Bill Gates. *The Road Ahead*. Viking, 1995.
- John Markoff. *What the Dormouse Said: How the 60s Counterculture Shaped the Personal Computer Industry*. Viking, 2005.

Chapter 9

- David Sheff. *Game Over: How Nintendo Conquered the World*. Vintage Books, 1993.

Chapter 10

- Jeff Hecht. *City of Light: The Story of Fiber Optics*. Oxford University Press, 1999.
- Stephen Segaller. *NERDS 2.0.1*. TV Books, 1998.
- Clay Shirky. *Here Comes Everybody*. Allen Lane, 2008.

Tom Standage. *The Victorian Internet: The Remarkable Story of the Telegraph and the Nineteenth Century's On-line Pioneers*. Walker and Company, 1998.

Jonathan Zittrain. *The Future of the Internet – And How to Stop It*. Allen Lane, 2008.

Chapter 11

danah boyd. *It's Complicated: The Social Lives of Networked Teens*. Yale University Press, 2014.

Amy Langville and Carl Meyer. *Google's PageRank and Beyond*. Princeton University Press, 2012.

Jaron Lanier. *You Are Not a Gadget*. Alfred A. Knopf, 2010.

David A. Vise. *The Google Story*. Delacourt Press, updated edition, 2008.

Chapter 12

Ross Anderson. *Security Engineering*. Wiley Publishing, 2nd edition, 2008.

Mark Bowden. *Worm: The First Digital World War*. Atlantic Monthly Press, 2011.

John Haynes and Harvey Klehr. *Venona: Decoding Soviet Espionage in America*. Yale University Press, 2000.

Jaron Lanier. *Who Owns the Future?* Simon & Schuster, 2013.

Stephen Roskill. *The Secret Capture*. Seaforth Publishing, 2011.

David Sanger. *Confront and Conceal: Obama's Secret Wars and Surprising Use of American Power*. Broadway Books, 2013.

Simon Singh. *The Code Book*. Fourth Estate Limited, 1999.

The RSA scheme is described in detail in an appendix of Singh's book.

Chapter 13

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2010.

Chapter 14

Stephen Baker. *Final Jeopardy*. Houghton Mifflin Harcourt, 2011.

Devinderjit Sivia and John Skilling. *Data Analysis: A Bayesian Tutorial*. Oxford University Press, 2nd edition, 2006.

Chapter 15

Eric Drexler. *Engines of Creation: The Coming Era of Nanotechnology*. Doubleday, 1986.

Sandy Fritz. *Understanding Nanotechnology*. Warner Books, 2002.

Gerard Milburn. *Quantum Entanglement and the Computing Revolution: The Quantum Processor*. Perseus Books, 1998.

Ed Regis. *Nano!: Remaking the World Atom by Atom*. Bantam Press, 1995.

Chapter 16

Lee Gutkind. *Almost Human: Making Robots Think*. W. W. Norton and Company, 2006.

Jeff Hawkins and Sarah Blakeslee. *On Intelligence*. St. Martin's Press, 2004.

Christof Koch. *Consciousness: Confessions of a Romantic Reductionist*. MIT Press, 2012.

Daniel Dennett. *Consciousness Explained*. Penguin Books, 1993.

Marvin Minsky. *The Society of Mind*. Simon & Schuster, 1987.

Chapter 17

Lois Gresh and Robert Weinberg. *The Computers of Star Trek*. Basic Books, 1999.

David Seed. *Science Fiction: A Very Short Introduction*. Oxford University Press, 2011.

David G. Stork (ed.). *Hal's Legacy: 2001's Computer as Dream and Reality*. MIT Press, 1997.

Patricia Warrick. *The Cybernetic Imagination in Science Fiction*. MIT Press, 1980.

Figure credits

The illustrations presented in this book are courtesy of the persons and organizations listed in this section. Any not listed here were drawn by the authors.

Chapter 1

- B.1.1:** Bloomsbury Publishing
B.1.2: © Los Alamos National Laboratory
B.1.3: © Science Photo Library
B.1.4: © Science Photo Library
B.1.5: Denis Gradel @tofz4u
B.1.6: Patrick Tresset and Frederic Fol Leymarie (Alkon-I project)
B.1.7: Jack Copeland
B.1.8: Wikimedia commons
B.1.9: Leo Computers Society
B.1.10: Horst Zuse
B.1.11: Sergiy Klymenko
B.1.12: Wayne Fitzsimmons and Commonwealth Scientific & Industrial Organisation
- Fig. 1.1:** Library Foundation, Buffalo and Erie County Public Library
Fig. 1.2: Mária and Gyula Pápay
Fig. 1.3: MIT Museum
Fig. 1.4: Paul W. Shaffer, SEAS University of Pennsylvania
Fig. 1.5: Fastfission/Wikimedia Commons
Fig. 1.6: Jan Van der Spiegel, University of Pennsylvania
Fig. 1.7: U.S. Army Photo, from 8 × 10 transparency, courtesy Harold Breaux, Times Mirror Magazines Inc., reprinted from *Popular Science* with permission © 1946
Fig. 1.8: Los Alamos National Laboratory/Science Photo Library
Fig. 1.9: Magyar Posta
Fig. 1.11: Antoine Taveneaux/London Science Museum
Fig. 1.12: London Evening News
Fig. 1.13: Wikimedia Commons

Fig. 1.14: School of Computer Science, University of Manchester

Fig. 1.15: No copyright holder found

Fig. 1.16: Royal Mail

Fig. 1.17: Science Museum/Science & Society Picture Library

Fig. 1.18: National Museum of Scotland/Ad Meskens

Fig. 1.19: Ian Taylor

Fig. 1.20: Ian Taylor

Fig. 1.21: Ian Taylor

Fig. 1.22: J. Lyons & Co. Archives

Cartoon: Szűcs Katalin Ágnes, published in Szűcs Ervin. A számítógép tegnaptól holnapig, Műszaki Könyvkiadó, Budapest, 1987, illustrations by Egri Béla

Timeline 1.1: Special Collections Department, Iowa State University Library

Timeline 1.2: Harvard University Archives, UAV 362.7295.8p, B 1, F 11, S 109

Timeline 1.3: Wikimedia Commons/KGGucwa

Timeline 1.4: LEO Computers Society

Timeline 1.5: Boris Malinovsky, ukrainiancomputing.info

Timeline 1.6: Division of Medicine & Science, National Museum of American History, Smithsonian Institution

Timeline 1.7: Horst Zuse

Timeline 1.8: U.S. Army Photo, from K. Kempf, "Historical Monograph: Electronic Computers within the Ordnance Corps"

Timeline 1.9: © Computer Laboratory, University of Cambridge

Timeline 1.10: Wikimedia Commons/Daderot

Timeline 1.11: U.S. Census Bureau

Timeline 1.12: Wikimedia Commons/John O'Neill

Chapter 2

- B.2.1:** MIT Museum

- B.2.2:** Wikimedia Commons

- B.2.3:** Wikimedia Commons/Sophia Elizabeth De Morgan
B.2.4: Jolanta Motylińska
B.2.5: Math & CS Dept., Denison University
B.2.6: Computer Laboratory, University of Cambridge
B.2.7: American Institute of Physics
B.2.8: MIT Museum
B.2.9: Photo used by permission of Birkbeck University of London
B.2.10: IBM Corporation

Fig. 2.4: Robert Szlizs

Fig. 2.6: Wikimedia Commons/Staecker

Fig. 2.7: Wikimedia Commons/Arne Nordmann

Fig. 2.15: Thomas Bressoud

Fig. 2.16: Danny Hillis

Fig. 2.17: Wikimedia Commons/Stacalusa

Fig. 2.20: Chris Nyberg, Tom Barclay, Zarka Cvetanovic, Jim Gray, and David B. Lomet: "AlphaSort: A Cache-Sensitive Parallel External Sort," *Very Large Data Bases Journal* no. 4 (1995): 608.

Fig. 2.22: School of Computer Science, University of Manchester

Fig. 2.23: David Link

Fig. 2.24: Wikimedia Commons/Konstantin Lanzet

Fig. 2.25: Science Museum/Science & Society Picture Library

Fig. 2.26: Wikimedia Commons/Vanderdecken

Cartoon: Luke Warm

Chapter 3

- B.3.1:** Michelle Feynman
B.3.2: Dr. Joyce Wheeler
B.3.3: Charis Tsevis
B.3.4: Harvard University Archives, UAV 362.7295.8p, B 1, F 11, S 109
B.3.5: IBM Corporation
B.3.6: Brian Randell
B.3.7: Chuck Painter/Stanford News Service
B.3.8: MIT Museum
B.3.9: IBM Corporation
B.3.10: Geoffrey Sharman

Fig. 3.1: Nikolai Rulkov

Fig. 3.2: Wikimedia Commons/Ricardo

Fig. 3.4: Joey Zanotti

Fig. 3.5: Naval Surface Warfare Center, Dahlgren, Virginia

Fig. 3.6: Unisys Corporation

Fig. 3.7: IBM Corporation

Fig. 3.8: Unisys Corporation

Fig. 3.9: Zazzle Inc.

Fig. 3.10: © The MITRE Corporation. All rights reserved.
Used with permission

Fig. 3.14: Roger Penwill

Fig. 3.15: IBM Corporation

Fig. 3.16: IBM United Kingdom Limited/Peter Facey

Cartoon: Szűcs Katalin Ágnes, published in Szűcs Ervin. A számítógép tegnaptól holnapig, Műszaki Könyvkiadó, Budapest, 1987, illustrations by Egri Béla

Chapter 4

- B.4.1:** IBM Corporation
B.4.2: Rutger M. Dijkstra
B.4.3: Rune Myhre
B.4.4: David Parnas
B.4.5: Bjarne Stroustrup
B.4.6: Wikimedia Commons/Peter Campbell
B.4.7: Grady Booch
B.4.8: Wikimedia Commons/Bill Ebbesen
B.4.9: Hal Abelson
B.4.10: Wikimedia Commons/Martin Streicher
B.4.11: Wikimedia Commons/Randal Schwartz
B.4.12: Wikimedia Commons/author unknown
B.4.13: Tony Hoare
B.4.14: IBM Corporation
B.4.15: Tony Hey
B.4.16: Erich Gamma

Fig. 4.1: No copyright holder found

Fig. 4.2: Chris Kania

Fig. 4.8: Grady Booch

Fig. 4.9: Panos Melas

Fig. 4.10: Courtesy of Microsoft Corporation

Fig. 4.11: Wikimedia Commons/Larry Ewing

Fig. 4.12: O'Reilly Media, Inc.

Fig. 4.13: Wikimedia Commons/Jules Mazur

Fig. 4.14: NASA Jet Propulsion Laboratory

Fig. 4.15: ESA/CNES

Fig. 4.16: NASA Jet Propulsion Laboratory/Corby Waste

Cartoon: © Paragon Innovations, used with permission.

Chapter 5

- B.5.1:** Wikimedia commons, postage stamp issued in the USSR in 1983
B.5.2: Stanford University
B.5.3: David Harel
B.5.4: Wikimedia commons, painting by Jakob Emanuel Handmann
B.5.5: Los Alamos National Laboratory
B.5.6: Stanford University
B.5.7: Stephen Cook
B.5.8: Leonid Levin
B.5.9: Richard Karp

Fig. 5.1: The Bodleian Library, University of Oxford, MS. Huntington 214, folio 1 recto

Fig. 5.7a: Wikimedia Commons/map modified by Bogdan Giusca

Fig. 5.7b: Wikimedia Commons/ author unknown

Fig. 5.7c: Wikimedia Commons/ author unknown

Fig. 5.8: Microsoft Corporation

Fig. 5.11: Communications of the ACM

Fig. 5.14: Reprinted by permission of Princeton University Press

Fig. 5.15: Reprinted by permission of Princeton University Press

Fig. 5.16: Robert Bosch

Fig. 5.18: David Harel

Fig. 5.19: David Harel

Cartoon: Luke Warm

Chapter 6

B.6.1: Robert Szilzs

B.6.2: No copyright holder found

B.6.3: Justin Weinmann

B.6.4: Oskar Morgenstern photographer. From the Shelby White and Leon Levy Archives Center, Institute for Advanced Study, Princeton, New Jersey

B.6.5: Patrick Tresset and Frederic Fol Leymarie (Alkon-I project)

B.6.6: Robert Szilzs

B.6.7: Alonzo Church Papers. Princeton University Archives. Department of Rare Books and Special Collections. Princeton University Library

Fig. 6.1: Scientific American

Fig. 6.2: Courtesy of Kevin Kelly

Fig. 6.3: Wikimedia Commons/Bill Bailey

Fig. 6.4: Wikimedia Commons/GabrielF

Fig. 6.5: Prentice-Hall/Marvin Minsky. *Computation: Finite and Infinite Machines*, 1967.

Fig. 6.6: Hal Abelson

Fig. 6.11: Records of the Office of the Director/Faculty Files/ Box 13/Kurt Gödel (Pre-1953 Memberships). From the Shelby White and Leon Levy Archives Center, Institute for Advanced Study, Princeton, New Jersey

Cartoon: American Scientist

Chapter 7

B.7.1: Penn State University Archives, Pennsylvania State University Libraries

B.7.2: Wikimedia Commons/AT&T

B.7.3: Wikimedia Commons/Robert Cathles

B.7.4: U.S. National Academy of Engineering

B.7.5: David Wayne Miller/Magnum Photos

B.7.6: SEMI (www.semi.org)

B.7.7: Wikimedia Commons/Intel Free Press

B.7.8: Robert Dennard

B.7.9: Marcian E. Hoff, Stan Mazor and Federico Faggin

B.7.10: Wikimedia Commons/Intel Free Press

B.7.11: Jon Brenneis

B.7.12: Charles Babbage Institute, University of Minnesota, Minneapolis

Fig. 7.1: Wikimedia Commons/ Bell Labs Holmdel

Fig. 7.2: Wikimedia Commons/NASA

Fig. 7.5: Reprinted with permission of Alcatel-Lucent USA Inc.

Fig. 7.7: DeGolyer Library, Collection number: A2005.0025, Texas Instruments Records

Fig. 7.8: Wikimedia Commons/Dick Lyon

Fig. 7.9: Jonathan Hey

Fig. 7.10: Fairchild Camera and Instrument Corporation

Fig. 7.11: U.S. Patent and Trademark Office

Fig. 7.12: Wikimedia Commons/U.S. Department of Defense

Fig. 7.13: Wikimedia Commons/Jnanna

Fig. 7.14: Tim McNeerney and Fred Huettig

Fig. 7.15a: © 1998 IEEE. Reprinted, with permission, from *Electronics* 38, no. 8 (19 April 1965): 114.

Fig. 7.15b: © Science photo library

Fig. 7.16: IBM Corporation

Fig. 7.17: Intel Corporation

Fig. 7.19: Intel Corporation

Fig. 7.23: Lawrence Livermore National Laboratory

Fig. 7.24: Geoffrey Fox

Cartoon: © 1998 IEEE. Reprinted, with permission, from *Electronics* 38, no. 8 (19 April 1965): 114.

Chapter 8

B.8.1: Microsoft Corporation

B.8.2: Forrest M. Mims III

B.8.3: Microsoft Corporation

B.8.4: Apple Corporation

B.8.5: Dan Bricklin and Bob Frankston © www.jimraycroft.com, 1982

B.8.6: IBM Corporation

B.8.7: Scott Kildall

B.8.8: Courtesy of Ilze Vanovska

B.8.9: Russian Space Agency

B.8.10: Microsoft Corporation

B.8.11: © Bill & Melinda Gates Foundation/Jeff Christensen

B.8.12: SRI

B.8.13: Gordon College and the Olsen family

B.8.14: C. Gordon Bell

B.8.15: Charles Thacker

B.8.16: Butler Lampson, author Bob McClure

B.8.17: Alan Kay

B.8.18: Geoffrey Nelson

B.8.19: IBM Corporation

B.8.20: IBM Corporation

Fig. 8.1: PARC, a Xerox company

Fig. 8.2: PARC, a Xerox company

Fig. 8.3: © Annie Leibovitz/Contact Press Images, the Artist

Fig. 8.4: Reprinted, with permission, from *Popular Electronics* magazine, January 1975

Fig. 8.5: Wikimedia Commons/Michael Holley

Fig. 8.6: Wikimedia Commons/Michael Holley

Fig. 8.7: Microsoft Corporation

Fig. 8.8: Computer History Museum

Fig. 8.9: Wikimedia Commons/Ed Uthman

Fig. 8.10: Wikimedia Commons/Michael Holley

Fig. 8.11: EAMule

Fig. 8.12: Wikimedia Commons

Fig. 8.13: IBM Corporation

Fig. 8.14: Wikimedia Commons/Alexander Schaelss

Fig. 8.15: IBM Corporation

Fig. 8.16: PARC, a Xerox company

Fig. 8.17: Bill Buxton

Fig. 8.18: Reprinted with permission from Blake Patterson

Fig. 8.19: SRI International

Fig. 8.20: SRI International

- Fig. 8.21:** PARC, a Xerox company
Fig. 8.22: Digital Equipment Corporation
Fig. 8.23: Wikimedia Commons/Joe Mabel
Fig. 8.24: Wikimedia Commons/Sascha Pohflepp
Fig. 8.25: Wikimedia Commons/Jeff Keyzer
Fig. 8.26: Wikimedia Commons/Dejdzer/Digga
Fig. 8.27: Dennis Sylvester, University of Michigan
Fig. 8.28: Buchlovský family
Fig. 8.29: © Apple Inc. Use with permission. All rights reserved. Apple and the Apple logo are registered trademarks of Apple Inc.

Cartoon: Roger Penwill

Chapter 9

- B.9.1:** No copyright holder found
B.9.2: Don Daglow
B.9.3: Nancy Crowther
B.9.4: Don Woods
B.9.5: Alissa Bushnell
B.9.6: spong.com
B.9.7: Toru Iwatani
B.9.8: Wikimedia Commons/Vincent Diamante
B.9.9: Wikimedia Commons/Official GDC photo
B.9.10: Wikimedia Commons/Official GDC photo (John Carmack) and Wikimedia Commons/John Romero
B.9.11: Minecraft Museum/Elin Zetterstrand

Fig. 9.1: No copyright holder found
Fig. 9.2a: MIT Museum
Fig. 9.2b: Wikimedia Commons
Fig. 9.3: No copyright holder found
Fig. 9.4: Wikimedia Commons/KaVir
Fig. 9.5: Wikimedia Commons/Rob Boudon
Fig. 9.6: Bandai Namco Games Inc.
Fig. 9.7: Microsoft Corporation
Fig. 9.8: Steven Collins
Fig. 9.9: © 1993, 1996, 2013 Cyan, Inc. All rights reserved. Used with permission
Fig. 9.10: Electronic Arts Inc.
Fig. 9.11: Oliver Robinson
Fig. 9.12: Zsolt Kósa
Fig. 9.13: Wikimedia Commons/Pierre Rudloff
Fig. 9.14: Wikimedia Commons/William Warby
Fig. 9.15: Destiny Andreen
Fig. 9.16: Wikimedia Commons/Juho Paavisto
Fig. 9.17: Stewart Brand
Fig. 9.18: Catholic University of Leeuven

Cartoon: Marty Bucella

Chapter 10

- B.10.1:** U.S. National Library of Medicine
B.10.2: Flickr/Yuichi Sakuraba
B.10.3: French Post
B.10.4a: Wikimedia Commons
B.10.4b: Smithsonian Institution Libraries
B.10.5: Wikimedia Commons/Jim Henderson
B.10.6: Bill Burns

- B.10.7:** RAND Corporation
B.10.8: © Science photo library
B.10.9: Lawrence Roberts
B.10.10: Robert Taylor
B.10.11: Len Kleinrock
B.10.12: © 2009 IEEE
B.10.13: Raytheon BBN Technologies
B.10.14: Elon University
B.10.15: Bob Metcalfe
B.10.16: Stanford University News Service/Jose Mercado
B.10.17: Wikimedia Commons/Carl Malamud
B.10.18: No copyright holder found
B.10.19: University of Southampton
B.10.20: No copyright holder found
B.10.21: Corning Incorporated
B.10.22: University of Southampton

Fig. 10.1: Wikimedia Commons/Zacatecnik
Fig. 10.2: Bill Burns/Atlantic-Cable.com
Fig. 10.3: Wikimedia Commons/Lokilech
Fig. 10.4: Library of Congress/Illustration: William Henry Jackson, American artist. Text: Howard Roscoe Driggs
Fig. 10.5: Wikimedia Commons/Andy Stephenson
Fig. 10.6: Wikimedia Commons from Louis Le Breton, témoin des marines du XIXème siècle
Fig. 10.7: Wikimedia Commons from Franz Eugen Köhler, Köhler's Medizinal-Pflanzen
Fig. 10.8: Wikimedia Commons/NASA
Fig. 10.9: Wikimedia Commons/David de Ugarte
Fig. 10.10: Robert Taylor
Fig. 10.11: Reprinted with permission of MIT Lincoln Laboratory, Lexington, Massachusetts
Fig. 10.12: Alexander McKenzie
Fig. 10.13: Robert Taylor
Fig. 10.14: Wikimedia Commons/Andrew Adams
Fig. 10.15: Wikimedia Commons/Yngvar
Fig. 10.16: SRI International
Fig. 10.18: Wikimedia Commons/Mhrmaw
Fig. 10.19: Patent(s) Pending and © Lumeta Corporation 2000–2013. All rights reserved. LUMETA, LUMETA Logo, IPSONAR and IPSONAR logo are trademarks and service marks of Lumeta Corporation
Fig. 10.20: © Berenice Abbott/Commerce Graphics Ltd., Inc., Courtesy Howard Greenberg Gallery, New York
Fig. 10.21: Wikimedia Commons/Timwether
Fig. 10.23: Wikimedia Commons/4028mdk09
Fig. 10.24: Wikimedia commons/Kevin Cordina
Fig. 10.25: Early Office Museum (officemuseum.com)

Cartoon: Robert Taylor

Chapter 11

- B.11.1:** MIT Museum
B.11.2: Ted Nelson
B.11.3: CERN
B.11.4: CERN
B.11.5: CERN
B.11.6: George Simian
B.11.7: Netscape Communications Corporation

- B.11.8:** Wikimedia Commons/James Duncan Davidson
B.11.9: Wikimedia Commons/Mitchell Aidelbaum
B.11.10: Wikimedia Commons/Joi Ito
B.11.11: Wikimedia Commons/Norbert Stuhrmann
B.11.12: Wikimedia Commons/Carrigg Photography
B.11.13: Wikimedia Commons/Kira Wales
B.11.14: Wikimedia Commons/Guillaume Paumier (guillaumepaumier.com)

- Fig. 11.1:** Life Magazine
Fig. 11.2: Ted Nelson
Fig. 11.3: Ted Nelson
Fig. 11.4: Wikimedia Commons/Diliff
Fig. 11.5: Wikimedia Commons/Maximilien Brice (CERN)
Fig. 11.6: CERN
Fig. 11.8: CERN
Fig. 11.10: SLAC
Fig. 11.11: Jim Halley and Silvano Gennaro
Fig. 11.12: Wikimedia Commons/Ragib Hasan
Fig. 11.13: CERN
Fig. 11.14: Wikimedia Commons/Netscape
Fig. 11.15: Wikimedia Commons/ICANN
Fig. 11.16: Wikimedia Commons/lizzielaroo
Fig. 11.17: Microsoft Corporation
Fig. 11.18: Estate of Gale Pitt
Fig. 11.20: Wikimedia Commons/Austin McKinley
Fig. 11.21: Linda A. Cicero/Stanford News Service
Fig. 11.22: Massimo Franceschet
Fig. 11.23: Wikimedia Commons/Steve Jurvetson
Fig. 11.24: Google, Inc./Connie Zhou
Fig. 11.25: Google, Inc./Connie Zhou
Fig. 11.27: Paweł Opole
Fig. 11.28: Twitter, Inc.
Fig. 11.29: Penelope C. Fahlman
Fig. 11.30: Wikimedia Commons/Jacob Bøtter

Cartoon: Luke Warm

Chapter 12

- B.12.1:** Clifford Stoll
B.12.2: Microsoft Corporation
B.12.3: Paul O. Boisvert
B.12.4: Wikimedia Commons/Trevor Blackwell
B.12.5: No copyright holder found
B.12.6: Mary Holzer and Martin Hellman
B.12.7: Ron Rivest
B.12.8: Phil Zimmermann
- Fig. 12.1:** Microsoft Corporation
Fig. 12.2: Clifford Stoll
Fig. 12.3: U.S. Government
Fig. 12.4: Wikimedia Commons/Avinash Meetoo
Fig. 12.5: Flickr/Intel Free Press
Fig. 12.7: Microsoft Corporation
Fig. 12.8: Robert Taylor
Fig. 12.9: Microsoft Corporation
Fig. 12.10: Wikimedia Commons/Jim Sanborn
Fig. 12.11: Wikimedia Commons/Hubert Berberich
Fig. 12.12: Reprinted by permission of Yale University Press

- Fig. 12.13:** Stephen W. Roskill
Fig. 12.16: No copyright holder found
Fig. 12.18: Natasa Milic-Frayling, Microsoft Research
Fig. 12.19: Wikimedia Commons/Ministry of Defence

Cartoon: Marty Bucella

Chapter 13

- B.13.1:** MIT Museum
B.13.2: Jonathan Michie
B.13.3: Luis von Ahn
B.13.4: Photo from the private collection of Prof. Dr. h.c. mult. Joseph Weizenbaum
B.13.5: Carnegie Mellon University
B.13.6: MIT Museum
B.13.7: Edward Feigenbaum
B.13.8: MIT Museum
B.13.9: Geoffrey Hinton/Photograph by Emma Hinton
- Fig. 13.2:** Hugh Loebner
Fig. 13.3: captcha.tv
Fig. 13.4: Tery Winograd
Fig. 13.5: Bruce Buchanan and Ted Shortliffe
Fig. 13.6: Wikimedia Commons/Carl Linnaeus
Fig. 13.7: Wikimedia Commons/DBpedia
Fig. 13.8: Wikimedia Commons/Agesscene
Fig. 13.10: Wikimedia Commons/Lotzman Katzman
Fig. 13.11: Wikimedia Commons/Jim Gardner
Fig. 13.12: Telegraph Media Group Limited 2001
Fig. 13.13: Wikimedia Commons/modified from the Gutenberg Encyclopedia
Fig. 13.14: Wikimedia Commons/Santiago Ramón y Cajal
Fig. 13.15: U.S. National Cancer Institute's Surveillance
Fig. 13.17: Wikimedia Commons

Cartoon: Robert Szlizs

Chapter 14

- B.14.1:** Soshichi Uchii
B.14.2: Jolanta Motylińska
B.14.3: Judea Pearl
B.14.4: MIT Museum
B.14.5: IBM Corporation
B.14.6: Robert Holmgren
- Fig. 14.1:** Wikimedia Commons/Ted Coles
Fig. 14.2: Judea Pearl
Fig. 14.3: Microsoft Research Cambridge/Jamie Shotton
Fig. 14.4: Microsoft Research Cambridge/Jamie Shotton
Fig. 14.5: Microsoft Research Cambridge/Jamie Shotton
Fig. 14.6: Microsoft Research Cambridge/Jamie Shotton
Fig. 14.7: Monika Papayova
Fig. 14.8: Microsoft Research Cambridge/Jamie Shotton
Fig. 14.9: Microsoft Research Cambridge/Jamie Shotton
Fig. 14.10a: Wikimedia Commons/Michelle Walz Eriksson
Fig. 14.10b: Wikimedia Commons/Logan Abassi/UNDP Global
Fig. 14.12: Wikimedia Commons/Clockready

Fig. 14.13: Wikimedia Commons/SethAllen623

Fig. 14.14: Jolyon Troscianko

Cartoon: Condé Nast Licensing/Tom Cheney

Chapter 15

B.15.1: IBM Corporation

B.15.2: K. E. Drexler

B.15.3: Chenming Hu

B.15.4: Wikimedia Commons/Campus Brasil Party

B.15.5: Harold Kroto

B.15.6: Sumio Iijima

B.15.7: University of Manchester

B.15.8: Stanley Williams

B.15.9: David Deutsch/Photo by Lulie Tanett

B.15.10: Wikimedia Commons/Peter Shor

B.15.11: Edward Fredkin

B.15.12: NIST/© Geoffrey Wheeler

B.15.13: Wikimedia Commons/Antony-22

B.15.14: Randy Rettberg, iGEM Foundation

B.15.15: Tom Knight

Fig. 15.1: Michelle Feynman

Fig. 15.2: Michelle Feynman

Fig. 15.3: Michelle Feynman

Fig. 15.4: IBM Corporation

Fig. 15.5: IBM Corporation

Fig. 15.6: IBM Corporation

Fig. 15.7: K. E. Drexler

Fig. 15.8: Sven Rogge

Fig. 15.9: Wikimedia Commons/Materialscientist

Fig. 15.10: Wikimedia Commons/Wing-Chi Poon

Fig. 15.11: IBM Corporation

Fig. 15.12: Wikimedia Commons/J. J. Yang, HP Labs

Fig. 15.13: J. E. Mooji

Fig. 15.15: Nicolas Gisin/Background image from NASA World Wind

Fig. 15.17: No copyright holder found

Fig. 15.18: J. Jost/NIST

Fig. 15.19: NIST

Fig. 15.20: NIST

Fig. 15.21: Wikimedia Commons

Fig. 15.22: © 2012 The Regents of the University of California. All rights reserved

Fig. 15.23: Tom Deerinck and Mark Ellisman of the National Center for Microscopy and Imaging Research at the University of California, San Diego

Fig. 15.24: Microsoft Corporation

Cartoon: David Simonds

Chapter 16

B.16.1: Wikimedia Commons/Christine Mary (Devol) Wardlow

B.16.2: Stevan Harnad

B.16.3: John Hopfield

B.16.4: Numenta, Inc.

B.16.5: Keith Weller/Johns Hopkins Medicine

B.16.6: Wikimedia Commons/Hayford Peirce/color corrected by Franks Valli

B.16.7: Wikimedia Commons/Romanpoet

Fig. 16.1: Wikimedia Commons/William Wardlow

Fig. 16.2: Wikimedia Commons/ICAPlants

Fig. 16.3: Wikimedia Commons/Alex North

Fig. 16.4: Wikimedia Commons/World Wide Gifts

Fig. 16.5: Wikimedia Commons/NASA/JPL-Caltech

Fig. 16.6: Wikimedia Commons/Mike Murphy

Fig. 16.7: Wikimedia Commons/United States Air Force

Fig. 16.8: Always Innovating, Inc.

Fig. 16.9: Wikimedia Commons/Zawersh

Fig. 16.10: “PCMCMerge” is Copyrighted to Blue Brain Project, EPFL, Lausanne, Switzerland

Fig. 16.11: Wikimedia Commons/Simon Viktória

Fig. 16.12: Jonathan Hey

Cartoon: Luke Warm

Chapter 17

B.17.1: Wikimedia Commons/Cover of “L’Algérie” Magazine

B.17.2: Wikimedia Commons Frederick Hollyer

B.17.3: Luxemburg Post

B.17.4: Marsh Library

B.17.5: NUVO News

B.17.6: Wikimedia Commons/Rowena Morrill

B.17.7: Subhankar Biswas

B.17.8: No copyright holder found

B.17.9: Roddenberry Entertainment Inc.

B.17.10: With thanks to the SK Film Archives LLC, Warner Brothers., the Kubrick family, and University of the Arts London

B.17.11: Památník národního písemnictví, Prague

B.17.12: Wikimedia Commons/Pete Welsch

B.17.13: Wikimedia Commons/John Johnson

Fig. 17.1: Wikimedia Commons/Helfmann

Fig. 17.2: Wikimedia Commons/Experimenter Publishing Company Inc. Hugo Gernsback

Fig. 17.3: Wikimedia Commons/David B. Spalging

Fig. 17.4: Boyce Duprey

Fig. 17.5: Sad Hill News

Fig. 17.6: Wikimedia Commons/Poster for Federal Theatre Project presentation of R.U.R. 17.7 Wikimedia Commons/A scene from R.U.R.

Fig. 17.7: Wikimedia Commons

Fig. 17.8a: Wikimedia Commons/N. Ricardo

Fig. 17.8b: Wikimedia Commons/Pam Peel

Fig. 17.9: Wikimedia Commons/Rik Morgan (www.handheld-museum.com)

Fig. 17.10: Flickr/b727260

Fig. 17.11: Stefan Imhoff

Fig. 17.12: “Brooklyn Terminator” by Peter Kokis, Brooklyn RobotWorks. Photo by Beth Brown

Fig. 17.13: Grégoire “Léon” GUILLEMIN

Fig. 17.14: Coren Macniven (corenmac.tumblr.com)

Fig. 17.15: James Stayte (www.stayteofheart.co.uk)

Fig. 17.16: Maxime Pecourt

- Fig. 17.17:** Photographer Chris Ridley © Grant Naylor Productions Ltd.
- Fig. 17.18:** Pan MacMillan UK, © Douglas Adams 1979
- Fig. 17.19:** Stephen Player
- Fig. 17.20:** Grand Central Publishing, a division of Hachette Book Group, Inc.
- Fig. 17.21:** Penguin Books Ltd.
- Fig. 17.22:** From *Timeline* by Michael Crichton, published by Cornerstone, reprinted by permission of The Random House Group Limited
- Fig. 17.23:** © www.spacechannel.org
- Fig. 17.24:** Ian Miller
- Fig. 17.25:** Courtesy of Summit Entertainment, LLC
- Fig. 17.26:** Rogue Synapse (<http://www.roguesynapse.com>)
- Fig. 17.27a:** Matthew Dupuis
- Fig. 17.27b:** No copyright holder found
- Fig. 17.28:** Hatchette Group

- Fig. 17.29:** Microsoft Corporation
- Fig. 17.30:** Microsoft Corporation

Epilogue

- E.1:** Maggie Jones

Appendix I

- A.1:** Wikimedia commons/Guillaume Paumier et al. (National Center for Biotechnology Information), Liquid_2003, Arne Nordmann & The Tango! Desktop Project

Appendix 2

- A.2:** *Continuing Innovation in Information Technology* © 2012, The National Academies Press

Name index

- Abelson, Hal, 73
Abramson, Norman, 210
Adams, Douglas, 345–346, 347
Adcock, Willis, 125–126
Addis, Louise, 227
Adleman, Len, 247, 256, 257, 311, 315–316
Aiken, Howard, 168
Alberti, Leon Battista, 253
Aldiss, Brian, 333, 343
al-Khowarizmi, Mohammad, 84, 85
Allard, James, 230–231
Allen, Fran, 172
Allen, Paul, 145, 147, 158
Amdahl, Eugene, 57
Anderson, Carl, 342
Anderson, Chris, 322–323
Anderson, Harlan, 164–165
Anderson, Sean, 235
Anderson, Tim, 176–177
Andressen, Marc, 224, 228, 259
Applegate, David, 96
Aristotle, 24, 269
Ashton, Kevin, 323
Asimov, Isaac, 268, 320, 336, 337, 338, 341–342, 343
Aspray, William, 202
Atalla, John, 127–128
Atanasoff, John Vincent, 16
Atkin, Larry, 272
Augarten, Stan, viii, 25, 165
- Babbage, Charles, xi, 8, 11–13, 18, 82, 84, 95, 115, 344, 345
Babbage, Henry, 13
Backus, John, 48, 50–51, 59, 67
Badham, John, 343
Baer, Ralph, 179–180
Bagley, Sarah, 219
Ball, Harvey, 241
Ballmer, Steve, 70
Baran, Paul, 197–201, 202–203
Bardeen, John, 123–124, 126
Barker, Ben, 205, 213
Barron, David, 76
Bartell, Bill, 205
Bayes, Thomas, 281–282
Bear, Greg, 347–348, 355
- Beard, Maston, 22
Bechtolsheim, Andy, 193, 236
Bell, Gordon, 166, 167
Bell, Ian, 185–186
Bendis, Brian, 182
Beranek, Leo, 204
Berners-Lee, Tim, 155, 220, 223, 224–228, 239, 241, 269–270
Bernoulli, Johann, 87
Bernstein, Alex, 271
Berry, Clifford, 16
Bethe, Hans, 88
Bezos, Jeff, 230, 231
Bigelow, Julian, 10, 264
Bina, Eric, 224, 228
Bixby, Robert, 96
Blake, Andrew, 289
Blank, Julius, 126
Blank, Marc, 176–177
Bloch, Felix, 121
Boggs, David, 143, 209, 210
Bokor, Jeff, 301–302
Bolt, Richard, 204
Booch, Grady, 69, 70
Boole, George, 24, 277
Booth, Andrew, 37
Bosch, Bob, 96, 97
Bowden, Mark, 250
Braben, David, 185–186
Bradbury, Norris, 5
Bradbury, Ray, 333
Bradley, David, 173
Brand, Stewart, 144, 191, 199
Brattain, William, 123–124
Bricklin, Dan, 150–151
Bride, Edward, 152
Brin, Sergey, 233–236
Bristow, Steve, 178
Brock, Rod, 153
Brooks, Eugene, III, 140
Brooks, Fred, 57, 58, 68, 171
Brown, Gordon, 105
Brunel, Isambard Kingdom, 197
Brunner, John, 248
Bryan, Roland, 213
Buchanan, Bruce, 268–269
Buffett, Warren, 158
- Bullock, Sandra, 353
Burks, Arthur, 10
Burrows, Michael, 233
Bush, Vannevar, 2–3, 4, 22, 23–24, 162, 166, 220–223
Bushnell, Nolan, 177–178, 180
Buxton, Bill, 159
Byron, Lord, 14, 15
- Cailliau, Robert, 224, 226, 227
Campbell, John W., Jr., 333, 336–337
Campbell, Murray, 273
Campbell-Kelly, Martin, 174, 202
Cantor, Georg, 108–109
Čapek, Josef, 341
Čapek, Karel, 320, 341, 342
Card, Orson Scott, 352–353
Carlin, John, 353–354
Carmack, John D., 186
Caselli, Giovanni, 336
Cerf, Vint, 193, 205, 206–207, 211–212, 213
Ceruzzi, Paul E., 144
Chang, Morris, 133–134
Chappe, Claude, 194–195
Chomsky, Noam, 291
Christiansen, Tom, 76
Christie, Julie, 338
Chua, Leon, 304
Church, Alonso, 111, 112
Churchill, Winston, 18, 19
Chvatal, Vasek, 96
Cirac, Ignacio, 312
Clark, Jim, 228, 259
Clark, Wesley, 141–142, 164, 182, 203, 213
Clarke, Arthur C., 82, 340
Clement, Joseph, 12, 13
Cocke, John, 171, 172
Cocks, Clifford, 262
Codd, Edgar “Ted,” 80
Cohen, Fred, 113, 247
Comrie, Leslie, 7–8
Condon, Joe, 272–273
Cook, Stephen, 99, 100
Cooke, William Fothergill, 195
Corbató, Fernando, 52, 53, 78, 164
Cornfield, Jerome, 285

- Cosell, Bernie, 205
 Cox, Richard, 282–284
 Cray, Seymour, 139, 140
 Crichton, Michael, 348–351
 Crick, Francis, 325, 331
 Crocker, Steve, 206, 207, 213
 Cronkite, Walter, 17
 Crowther, William, 176, 205,
 207–208
 Cruise, Tom, 344
 Cunningham, Ward, 237–238
 Curl, Robert, 303–304
 Cutler, Dave, 166
 Dabney, Ted, 177–178
 Daglow, Don, 175–176
 Dahl, Ole-Johan, 65
 Daniels, Bruce, 176–177
 Daniels, Cal, 222–223
 Dantzig, George, 95, 96
 Davidoff, Monte, 146–147
 Davies, Donald, 200, 201, 202–203,
 211–212
 Dawkins, Richard, 347
 De Morgan, Augustus, 14–15, 24, 25
 Dean, Mark, 154
 Deng, Li, 277
 Dennard, Robert, 129, 130, 132
 Dennet, Daniel, 330, 331
 Dertouzos, Michael, 228
 Descartes, René, 325
 Deutsch, David, 306, 308, 349
 Deutsch, Peter, 162
 Devol, George, 320
 Dick, Philip K., 343–344, 345
 Diffie, Whitfield, 254, 255, 262
 Dijkstra, Edsger, 41, 58, 63, 69, 79,
 93–94, 145
 DiNucci, Darcy, 237
 Dirac, Paul, 137
 Dong Yu, 277
 Douglas, Alexander, 174
 Drexler, Eric, 299–300, 301, 348
 Dummer, Geoffrey, 124–125
 Dupuis, Matt, 353
 Eckert, J. Presper, 3–4, 5–7,
 9–10, 36, 48
 Edwards, Dan, 175
 Eigler, Don, 299, 300
 Einstein, Albert, 104, 137, 309
 Eisenhower, Dwight D., 18
 Elliot, John, 338
 Ellis, James, 262
 Elő, Árpád, 272
 Engelbart, Doug, 161, 162, 163, 191, 203,
 207, 213, 222, 229
 English, Bill, 162, 163
 Erickson, Jim, 173
 Ericsson, 159
 Estridge, Don, 152
 Euclid, 85
 Euler, Leonhard, 86, 87, 90–92, 99
 Euler, Paul, 87
 Evans, David, 161, 178, 182, 184
 Faggin, Frederico, 130, 131, 170
 Fahlman, Scott, 241
 Fairclough, John, 57
 Farr, William, 12–13
 Feigenbaum, Ed, 268–269
 Feistel, Horst, 254
 Felten, Ed, 96
 Ferrucci, David, 294–296
 Fetter, William, 182
 Feynman, Richard, 1–2, 44, 87, 138,
 298–299, 305–306, 315
 Filo, David, 232, 233, 236
 Fitzgerald, Zelda, 181
 Fitzgibbon, Andrew, 289
 Flaherty, Paul, 233, 235
 Fleming, Ian, 253–254
 Flowers, Tommy, 18, 19, 253, 262
 Forrester, Jay, 36–37, 164
 Forster, E. M., 334–335
 Fox, Geoffrey, 140
 foy, brian d., 76
 Frankston, Robert, 150–151
 Fredkin, Ed, 267, 272–273, 307–308
 Frege, Gottlob, 103
 Friedman, Harry, 295, 296
 Fuchs, Klaus, 253, 254
 Fulkerson, Ray, 95, 96
 Furber, Steve, 171–172
 Gambling, Alec, 214–217
 Gamma, Erich, 81
 Gardner, Martin, 257
 Gates, Bill, 79, 145–147, 148, 152, 153,
 156–158, 165, 175, 231–232
 Gates, Melinda, 158
 Geim, Andre, 304
 Geisman, Jim, 205
 Gelfand, Alan, 286
 Gernsback, Hugo, 334, 336
 Gibson, William, 351–352, 355
 Gill, Stanley, 46, 47
 Gingrich, Victor, 126
 Gödel, Kurt, 104, 117, 326–327
 Gold, Tommy, 36
 Goldfarb, Charles, 241
 Goldstine, Herman, 3–4, 6, 10, 62
 Golgi, Camillo, 274
 Good, Jack, 285
 Gosling, James, 67
 Gould, Benjamin, 12
 Gower, Andrew, 186
 Gower, Paul, 186
 Graetz, Martin, 174–175
 Gray, Jim, 32, 80
 Greenberg, Bob, 147
 Greenblatt, Richard, 272
 Griffin, Merv, 293–294
 Grotschel, Martin, 96
 Grove, Andy, 128
 Grover, Lov, 311
 Guevara, Che, 253
 Hafner, Katie, 205–206
 Hamilton, William, 99
 Hamming, Richard, 206
 Hanson, Rowland, 157
 Harel, David, 69, 70, 84–85, 86, 100, 112
 Harnad, Stevan, 326
 Harrison, Bill, 179–180
 Harrison, Harry, 355–357
 Hauser, Herman, 171–172
 Hawkins, Jeff, 327–328, 329
 Heart, Frank, 205–206, 207–208, 213
 Heath, Jim, 303
 Heckerman, David, 286–287
 Hed, Mikael, 188–189
 Hed, Niklas, 188–189
 Heiman, Frederic, 127–128
 Heisenberg, Werner, 137
 Held, Michael, 96, 98
 Hellman, Martin, 254, 255, 262
 Helm, Richard, 81
 Helsgaun, Keld, 96
 Hennessy, John, 171
 Herbert, Frank, 185
 Herschel, John, 11
 Herzfeld, Charles, 202, 213, 219
 Hess, Markus, 245–246
 Hey, Tony, viii, 238
 Higinbotham, William, 178
 Hilbert, David, 4, 102–106, 108, 110,
 111, 113
 Hillis, Danny, 23, 30
 Hiltzik, Michael, 142–143, 161, 167–168
 Hinton, Geoffrey, 276, 277
 Hippocrates, 273
 Hoare, Tony, 58, 79, 82
 Hockham, George, 214–216
 Hockney, Roger, 139
 Hodges, Andrew, 104, 105, 106
 Hoefer, Don, 121
 Hoeneisen, Bruce, 132
 Hoerni, Jean, 126, 127, 128
 Hoff, Ted, 130
 Hofstein, Steven, 127–128
 Hone, Joseph, 273
 Hopfield, John, 327
 Hopper, Grace, 47–49, 50–51
 Horn, Paul, 293–294
 Hoyle, Fred, 338
 Hsu, Feng-hsiung, 273
 Hu, Chenming, 301–302
 Huang, Jen-Hsun, 134
 Hung, Terry, 227
 Huygens, Christiaan, 137
 Hyponnen, Mikko, 247–248
 Iijima, Sumio, 303
 Iisalo, Jaako, 189
 Iklé, Fred, 285
 Iles, Greg, 358
 Ingalls, Dan, 169
 Isaacson, Walter, 157–158
 Iwatani, Toru, 179, 180
 Jacobson, Ivar, 69, 70
 Jacquard, Joseph-Marie, 14, 15
 Javey, Ali, 301
 Jeffreys, Harold, 284
 Jennings, Ken, 294, 295, 296

- Jobs, Steve, 149–150, 154–156, 157–158, 160, 178
 Johnson, E. A., 159
 Johnson, Ralph, 81
 Johnson, Selmer, 95, 96
 Johnson, Steven, 72, 78
 Jones, Michael, 287–288
 Joy, Bill, 79, 193, 212–213, 299–300
- Kahn, Bob, 193, 205, 206, 207–208, 209, 211–212, 213
 Kaku, Michio, 302
 Kao, Charles, 214–216, 217
 Karp, Richard, 96, 98, 100
 Kasparov, Garry, 273
 Kay, Alan, 66, 144, 163, 167–170, 191
 Keck, Donald, 216, 217
 Keen, Harold, 284–285
 Kemény, John, 145
 Kernighan, Brian, 78, 79, 96, 249
 Khang, Dawon, 127–128
 Khosla, Vinod, 193
 Kidder, Tracy, viii
 Kilburn, Tom, 10, 35, 36
 Kilby, Jack, 125–126
 Kildall, Gary, 152–153, 173
 Kipman, Alex, 288
 Kirstein, Peter, 212
 Kitaev, Alexei, 313
 Kleene, Stephen, 111
 Kleiner, Eugene, 126
 Kleinrock, Len, 203, 206, 213
 Kline, Charley, 206
 Knight, Tom, 313
 Knuth, Donald, 84–85, 86, 171
 Koch, Christof, 331
 Kompfner, Rudolf, 214
 Kotok, Alan, 272
 Kroto, Harry, 303–304
 Kubrick, Stanley, 268, 340
 Kunz, Paul, 227
 Kurtz, Thomas, 145
- Lampson, Butler, 39, 133, 162, 163, 167–170, 210, 318, 319
 Lampson, Lois, 167
 Lane, Jim, 147
 Laplace, Pierre-Simon, 282
 Lardner, Dionysius, 12
 Last, Jay, 126
 Lebedev, Sergei, 21–22
 Lebling, Dave, 176–177
 Lederberg, Joshua, 268–269
 Lehovec, Kurt, 126, 128
 Leibniz, Gottfried Wilhelm, 26
 Leinster, Murray, 338–339
 LeMay, Curtis, 285
 Lenat, Douglas, 269
 Letwin, Gordon, 147
 Levin, Leonid, 100
 Levy, David, 272
 Lewis, Andrea, 147
 Lickel, Charles, 293
 Licklider, J. C. R., 53, 78, 141, 142, 161–162, 163, 192–193, 201, 204–205, 208, 359
 Liebowitz, Annie, 191
- Lin, Shen, 96
 Linnaeus, Carolus, 269, 270
 Liu, Tsu-Jae King, 301–302
 Loebner, Hugh, 266
 Lorie, Raymond, 241
 Loth, Sebastian, 299–300
 Lovelace, Ada, 14–15
 Lowe, Bill, 151–152
 Lubow, Miriam, 147
 Lucas, George, 342
 Lyon, Matthew, 205–206
- Markkula, Mike, 150
 Markov, Andrei, 286
 Martin, Oliver, 96
 Mauchly, John, 3–4, 5–7, 9–10, 36, 48
 Maurer, Bob, 216, 217
 Mayfield, Mike, 175
 Mazor, Stan, 130
 McAllester, David, 270
 McCarthy, John, 52–53, 60, 63–64, 78, 142, 146, 164, 266, 267, 268, 272, 324
 McCollough, Peter, 163
 McConnell, Steve, 71
 McCulloch, Warren, 198, 273, 274, 275–276
 McDonald, Marc, 147
 McGrayne, Sharon, 284
 McGregor, Scott, 157
 McKenzie, Alex, 203
 McLellan, Bill, 298, 299
 McNealy, Scott, 193
 Mead, Carver, 120, 131–132
 Meier, Sid, 184
 Menabrea, Luigi, 14, 15
 Mensch, Bill, 171–172
 Merkle, Ralph, 262
 Metcalfe, Bob, 143, 209–210
 Metropolis, Nicholas, 87
 Michie, Donald, 264, 270
 Miller, Rand, 185
 Miller, Robin, 185
 Minsky, Marvin, 111, 266, 268, 355–357, 359
 Mitchell, Billy, 179
 Miyamoto, Shigeru, 174, 180–181
 Monier, Louis, 233
 Montgomery, Bernard, 18
 Montulli, Lou, 260
 Moore, Fred, 149
 Moore, Gordon, 126, 128, 129, 131–133, 298, 300, 301
 Morgan, Thomas Hunt, 270
 Morris, Bob, Jr., 248–249
 Morris, Bob, Sr., 248
 Morse, Samuel, 195–196, 219
 Mosher, Edward, 241
 Mott, Tim, 169
 Motwani, Rajeev, 234
 Mountcastle, Vernon, 328, 329
 Moynahan, Bridget, 343
- Naur, Peter, 51
 Naylor, Bill, 213
 Nelson, Ted, 222–223, 227
 Newell, Allen, 1, 166, 263, 266–267, 272
 Newman, Max, 18, 105–106
- Newman, Robert, 204
 Newman, Tom, 299
 Newton, Darwin, 169
 Newton, Isaac, 137
 Niknejad, Ali, 301
 Nishikado, Tomohiro, 178
 Noble, David, 150
 Norvig, Peter, 324–325
 Novoselov, Konstantin, 304
 Noyce, Robert, 125, 126, 128, 129, 131
 Nygaard, Kristen, 65
- O'Brien, Sean, 303
 Ohl, Russell, 123
 Olsen, Ken, 141–142, 164–165, 166–167
 Olsen, Stan, 164–165
 O'Rear, Bob, 147
 O'Reilly, Tim, 237
 Ornstein, Severo, 205, 213
 Orwant, Jon, 76
 Orwell, George, 155, 173
 Osborne, Adam, 170
 Otama, Terry, 156
 Otto, Steve, 96
 Ozzie, Ray, 324
- Padberg, Manfred, 96
 Page, Larry, 233–236
 Pajitnov, Alexey, 188
 Papert, Seymour, 276
 Parnas, David, 66, 68
 Paterson, Tim, 153, 173
 Patterson, David, 171
 Pauli, Wolfgang, 121, 138
 Payne, David, 216–218
 Pearcey, Trevor, 22
 Pearl, Judea, 285, 286
 Pellow, Nicola, 227
 Pender, Howard, 9
 Penrose, Roger, 326–327
 Perlis, Alan, 1, 33
 Persson, Markus, 186–187
 Pinkerton, John, 20
 Pitts, Walter, 198, 273, 274, 275–276
 Plummer, James, 133–134
 Pnueli, Amir, 69
 Poe, Edgar Allan, 336
 Polly, Jean Armour, 224
 Polya, George, 4
 Post, Emil, 111
 Postel, Jon, 206, 207, 212, 213
 Pouzin, Louis, 211–212
 Pratchett, Terry, 346–347
 Price, William, 281
 Prinz, Dietrich, 271
 Proyas, Alex, 343
 Putnam, Hilary, 325
- Raiffa, Howard, 285–286
 Ramón y Cajal, Santiago, 274, 275
 Raskin, Jef, 155
 Raymond, Eric, 59
 Redford, Robert, 351
 Rettberg, Randy, 314
 Richards, Martin, 78

- Rimmer, Peggy, 226
 Ritchie, Dennis, 67, 72, 78, 79, 249
 Rivest, Ron, 256, 257, 258–259, 311
 Roberts, Ed, 143–144, 145, 146
 Roberts, Frank, 216
 Roberts, Larry, 192–193, 201, 202–204, 208, 213
 Roberts, Sheldon, 126
 Rochester, Nathaniel, 266
 Rock, Arthur, 133
 Roddenberry, Gene, 339
 Rogers, Carl, 279
 Romero, John, 186
 Rosenblatt, Frank, 275–276
 Rosenblueth, Arturo, 264
 Rumbaugh, James, 69, 70
 Rumelhart, David, 276
 Russell, Bertrand, 103, 266–267
 Russell, Steve, 146, 174–175, 191
 Russell, Stuart, 324–325
 Russinovich, Mark, 247–248, 354–355
 Rutter, Brad, 295, 296
- Saarinen, Eero, 120
 Sagan, Carl, 268
 Sams, Jack, 152
 Samson, Peter, 175
 Sanger, David E., 243, 252
 Scantlebury, Roger, 201, 203, 213
 Scheutz, Edvard, 12, 13
 Scheutz, George, 12, 13
 Schlaifer, Robert, 285–286
 Schleiden, Matthias, 274
 Schmidt, Eric, 234
 Schröedinger, Erwin, 137, 309
 Schultz, Peter, 216, 217
 Schwann, Theodor, 274
 Schwarzenegger, Arnold, 345
 Schweizer, Erhard, 300
 Scott, Ridley, 155, 344, 345
 Sculley, John, 155–156
 Searle, John, 296–297, 325
 Seeman, Nadrian, 313
 Seitz, Chuck, 140
 Sendall, Mike, 225–226
 Shamir, Adi, 256, 257, 311
 Shannon, Claude, 23–25, 27, 171, 266, 267, 270–271
 Shelley, Mary, 333
 Sherrington, Charles, 275
 Shima, Masatoshi, 130
 Shoch, John, 248
 Shockley, William, 123–124, 126
 Shor, Peter, 306, 308, 311, 313
 Shortliffe, Edward, 269
 Shotton, Jaime, 288–290
 Shugart, Alan, 150
 Shum, Harry, 237
 Simmons, John, 20
 Simon, Herbert, 1, 263, 266–267, 272
 Simonyi, Charles, 157, 168–169
- Singh, Simon, 254, 262
 Sinofsky, Steven, 231
 Skinner, B. F., 325
 Slate, David, 272
 Slater, Robert, 151
 Smalley, Richard, 303–304
 Smith, Adrian, 286
 Smith, E. E., 174–175
 Smith, Will, 343
 Smith, William, 1–2
 Solomon, Les, 144
 Speilberg, Steven, 343
 Spiegelhalter, David, 286
 Stallman, Richard, 72, 73
 Standage, Tom, 193–194
 Starkweather, Gary, 209
 Steane, Andrew, 313
 Stephenson, Neal, 348, 355
 Sterling, Bruce, 351–352
 Stibitz, George, 30
 Stoll, Clifford, 245, 248
 Stross, Charles, 351
 Stroustrup, Bjarne, 41, 67
 Suarez, Daniel, 343
 Subramanian, Vivek, 301
 Sussman, Gerry, viii
 Sutherland, Ivan, 161, 163, 171, 178, 182, 184
 Swade, Doron, 12, 13, 22
 Szalay, Alex, 80
- Tananbaum, Andy, 73
 Tattam, Peter, 230–231
 Taylor, Bob, 142, 144, 162, 163, 169, 170, 191, 201–202, 208, 209–210, 213, 219
 Teller, Edward, 5, 87, 88
 Tesler, Larry, 155, 169
 Thacker, Chuck, 133, 163, 167–170, 209–210
 Thacker, Karen, 167
 Thatch, Truett, 205, 213
 Thompson, Ken, 72, 78, 79, 272–273
 Thomson, William, 196
 Thrope, Marty, 205
 Thrun, Sebastian, 322
 Tolkien, J. R. R., 176
 Tomlinson, Ray, 208–209
 Torvalds, Linus, 59, 73–75
 Traub, J. F., 84–85
 Trubshaw, Roy, 177, 186
 Tunnicliffe, William, 241
 Turing, Alan, 2, 8–9, 18, 102, 105, 107–110, 111, 112, 253, 270, 275, 284–285, 359
- Ulam, Stanislaw, 86–87, 88, 271
- van Dam, Andy, 226
 Vernam, Gilbert, 253
 Verne, Jules, 333, 335–336
 Viola, Paul, 287–288
 Vise, David, 234, 236
- Vlissides, John, 81
 von Ahm, Luis, 266
 von Neumann, John, x, 3–7, 9, 10, 33, 37, 62, 87, 88, 89–90, 104, 171, 273, 274
 Vonnegut, Kurt, 337–338
- Walden, Dave, 205, 207–208, 213
 Wales, Jimmy, 238
 Wall, Larry, 76
 Wallace, Bob, 147
 Wallace, James, 173
 Watson, John, 325
 Watson, Tom, Jr., 38, 57
 Watson, Tom, Sr., 38
 Weaver, Warren, 280
 Weber, Steven, 74–75
 Weiland, Ric, 147
 Weizenbaum, Joseph, 267, 279
 Welchman, Gordon, 284–285
 Wells, H. G., 333–334, 335, 336
 Wessler, Barry, 213
 Wheatstone, Charles, 15, 195, 336
 Wheeler, David, 45–46, 47
 Whitehead, Alfred, 103, 266–267
 Whitehouse, Edward, 196
 Whitney, Albert, 284
 Whittaker, Red, 322
 Whitworth, Joseph, 12
 Wiener, Norbert, 263, 264, 273
 Wilkes, Maurice, x, 7–8, 9, 10, 20, 35–36, 41, 45, 46, 47, 57
 Williams, David, 225–226, 227
 Williams, Freddie, 10, 35, 36
 Williams, Ronald, 276
 Williams, Samuel, 30
 Williams, Stan, 304–305
 Williamson, Malcolm, 262
 Wilson, Daniel, 343, 344
 Wilson, Sophie, 171–172
 Wineland, David, 312, 313
 Wing, Jeannette, ix
 Wingfield, Mike, 207
 Winograd, Terry, 268, 291–292
 Wirth, Niklaus, 79
 Womersley, J. R., 8–9
 Wood, Marla, 147
 Wood, Steve, 147
 Woods, Don, 176, 177
 Wozniak, Stephen, 148–150, 151, 155, 178
 Wyndham, John, 341
- Yang, Jerry, 232, 233
 Young, Thomas, 137
- Zepler, Eric, 215
 Zimmermann, Phil, 257–259
 Zoller, Peter, 312
 Zuckerberg, Mark, 239
 Zuse, Konrad, x, 21, 22

General index

- 16-bit microprocessors, 152, 153, 154, 166
- abacus, 26
- abstraction, 1–2, 6–7, 23, 27, 28, 30, 65–66, 80, 81
- Acorn Computers Ltd., 171–172, 185–186
- ADA programming language, 82
- address space, 55
- Advanced RISC Machines (ARM) Holdings, 171–172
- agents, 324–325
- AIBO 321
- Alcorn, Al, 178
- Algol 60 programming language, 51
- algorithms, 84–101
- alpha-beta pruning, 272
 - computability, 100–101
 - Euclid's algorithm, 85
 - greatest common divisor (GCD), 85
 - greedy algorithms, 92
 - numerical, 86–88
 - discrete approximation, 86
 - Monte Carlo methods, 86–88
 - pseudo-random numbers, 88
 - sorting, 88–90, 98
 - bubble sort, 89, 90, 96–97
 - merge sort, 89–90, 91, 96–97
 - minimax algorithm, 272
 - simplex algorithm, 95
- Altair 8800, 143–145, 146, 147, 148, 158
- Alto, 133, 156, 167–170, 209–210, 248
- ALU (arithmetic logical unit), 6, 7, 33–34
- Amazon, 230, 231, 319
- Analytical Engine, 13–14, 15
- AND operations, 24, 25, 27, 29, 307–308
- API (Application Programming Interface), 230–231
- Apple, 149–150, 154–157, 160, 340–341
- Application Programming Interfaces (APIs), 230–231
- Arecibo radio telescope, 28
- Ariane 5 rocket, 82, 83
- arithmetical logical units (ALUs), 6, 7, 33–34
- ARM (Advanced RISC Machines) Holdings, 171–172
- ARPA (Advanced Research Projects Agency), 79, 142, 161–162
- ARPANET, 61, 142, 176, 192, 193, 201–208, 211, 219, 229, 243–244
- artificial intelligence (AI), 161, 266, 355–358
 - Advice Taker program, 267–268
 - chatbot programs, 265, 266, 279
 - cognitive science, 264
 - computer chess, 270–273
 - cybernetics, 263, 264
 - expert systems, 268–269
 - feedback loops, 263–264
 - Logic Theorist program, 266–267
 - ontology, 269–270
- ASCII (American Standard Code for Information Interchange), 89
- ASIMO (Advanced Step in Innovative Mobility) robot, 321
- assemblers, 46
- assembly languages, 45–46, 59
- assignment, 21, 61
- Atari, 149, 178, 180, 183
- back-propagation networks, 276–277, 327
- band theory, 121–122
 - conduction band, 122
 - conduction electrons, 120
 - energy bands, 121, 122
 - valence band, 122
 - valence electrons, 122
- barber's paradox, 103, 104
- BASIC, 63, 145, 146–147, 148, 153
- batch processing, 51–52, 164
- Bayes Rule, 281, 283–284
- Bayesian inference, 281–287
 - Gibbs sampling, 286
 - likelihood function, 284
 - Markov chains, 286
 - Monte Carlo method, 286
 - priors (prior beliefs), 280–281
 - posterior beliefs, 281
 - sum and product rules, 283
 - thought experiments, 281–282
- Bayesian networks, 285, 286
- BBC Microcomputer, 172, 185–186
- BBN (Bolt, Beranek, and Newman), 204–208
- BCPL programming language, 78
- Bell Labs, 51, 67, 78, 120, 123–124, 127–128, 214, 216, 217, 291
- Bell's Law, 172
- binary arithmetic, 6–7, 22, 25–27, 28
- BIOS (Basic Input/Output System), 153
- bitmaps, 168, 183
- bits, 9, 19, 24–30, 35–36, 53, 126, 130, 149, 163
- Bletchley Park, 18–19, 88, 105, 253, 254, 262, 266, 284–285
- blogs, 237, 238
- bombe machines, 18, 19, 264, 284–285
- Boolean, 24–25, 60
- botnets, 244, 250–251
- brain, 273–275, 328, 329
- Brain virus, 246–247
- Bravo text editor, 157, 167, 168–169
- Bugs, 48, 70–72
- bus architecture, 144, 166
- bytes, 27
- C programming language, 67, 72, 78, 79, 249
- C++ programming language, 67
- CA (central arithmetic unit), 6, 7
- CAD (computer-aided design), 184, 187
- calculators, 129–130, 144
- CAPTCHAs, 266
- carbon, 302–304, 305
- CERN, 224–228
- chemical vapor deposition (CVD), 215, 216–217
- chess, 270–273
- Chinese Room, 296–297, 325
- Church-Turing thesis, 2, 111, 112
- CISC architecture, 171
- clocks, 31, 33
- cloud computing, 324
- CMOS, 127–128, 133, 140, 301
- CNOT gates, 307–308, 310
- coaxial cables, 210
- COBOL programming language, 47, 50, 51, 59, 63
- cognition, 325–326
- cognitive science, 264
- Colossus machine, 18–19, 88, 253, 262
- combinational circuits, 31
- command-line interface, 191
- compilers, 48–51
- complexity theory, 96–98
 - big-O notation, 97

- complexity theory (*cont.*)
 exponential time, intractable problems, 98, 99, 100
 NP-complete problems, 98–100
 polynomial time, tractable problems, 97–100
- computable numbers, 107–110, 111
 computer science, 1, 47, 161–162
 computer vision, 287–290, 291
 computers, 336–341
 conditional jumps, 44, 47
 Control-Alt-Delete combination, 173
 control statements, 61
 control unit (CU), 6, 33–34
 cookies, 259–260
 Corning Glass Works, 216, 217
 Cosmic Cube parallel computer, 140
 CP/M operating system, 152–153, 170, 173
 Cray, 139–140
 crowdsourcing, 59, 269–270, 292, 319
 cryptography, 46, 228–229, 247, 252–259, 262
 asymmetric versus symmetric encryption, 256
 cookies, 260
 Data Encryption Standard (DES), 254
 Diffie-Hellman key exchange, 254–256
 one-way functions, 254–256
 Pretty Good Privacy, 257–259
 public-key cryptography, 256–257
 RSA encryption, 256–257
 secret-key encryption, 253–254
 secure sockets layer (SSL), 228–229, 259
 shift ciphers, 252–253
 crystal lattice structure, 120–121
 cyberterrorism, 353–355
- dark silicon, 135
 DARPA Grand Challenge, 322
 data abstraction, 65–66
 data types, 60, 65–66
 databases, 79–80
 datagrams, 211
 DBPedia project, 269–270
 DEC (Digital Equipment Corporation), 145, 147, 164–165, 166–167, 174–175, 210–211
 declarative programming languages, 60, 63
 Deep Blue computer, 273, 296
 Deep Thought computer, 273
 DENDRAL program, 268–269
 denial of service attacks, 250
 DEUCE computer, 166, 200
 diagonal slash procedure, 109, 110
 Difference Engine, 11–13, 351–352
 Differential Analyzer, 2–3, 22, 23–24, 221
 digital certificates, 259
 digital signature authentication, 258
 directed acyclic graphs (DAGs), 93, 286
 distributed networks, 198–200, 202–203
 DNA computing, 313, 314–316
 do loops, 61
 domain names, 229–230
 doping, 122–123
 dot-com bubble, 230, 242
- DRAM (dynamic random access memory), 129, 130, 133
 drones (unmanned aerial vehicles), 322–323
 Dynabook concept, 144, 163, 168
 dynamic data structures, 59, 64–65
- e-commerce, 228–229
 EDFAs (erbium doped fiber amplifiers), 217
 EDSAC (Electronic Delay Storage Automatic Calculator), xi, 9–10, 22, 35–36, 45, 46, 174
 EDVAC (Electronic Discrete Variable Computer), 3, 5–10, 35–36
 ELIZA program, 265, 279
 email, 208–209, 243–244, 261
 emoticons, 241
 ENIAC (Electronic Numerical Integrator And Computer), 3–5, 6, 318
 Enigma machine, 18, 19, 253–254, 284–285
Entscheidungsproblem, 104, 105–106, 110, 111, 113
 Ethernet, 143, 167, 209, 210–211
 exclusion principle, 121, 138
 expert systems, 268–269
- Facebook, 238, 239, 319
 Fairchild Semiconductor, 126, 127, 128–129
 feedback loops, 263–264
 Ferranti Mark I computer, 35, 271
 fetch-execute cycle, 23, 44
 fiber-optic technology, 213–218
 FIFO (First In, First Out) stack, 65
 File Allocation Table (FAT) file system, 173
 file clerk model, 41–44
 flash memory, 304
 flip-flop, 31, 32
 floating-point data type, 65–66
 floppy disks, 150, 246
 flowcharts, 62, 69, 85
 FLOW-MATIC language, 50–51
 FORTRAN, 49, 50, 51, 59–61, 63, 65–66, 86
 Free Software Foundation, 72, 73
 frequentist, 280, 281, 286
 FSMs (Finite State Machines), 116
 FTP (File Transfer Protocol), 207
 functional abstraction, 23, 27, 28, 30
 fuzz testing, 69–70
- game tree, 270–272
 games and gaming, 174–189
 Angry Birds, 188–189
 chess, 270–273
 Civilization, 184
 Colossal Cave Adventure, 176, 177
 Doom, 186
 Dune II, 185
 Dungeon, 175–176
 Elite, 185–186
 FarmVille, 189
 FPS (first-person shooter) games, 186
 Halo game, 181–182, 357–358
 Happy Farm game, 188–189
 Minecraft, 186–188
- MMORPGs (massively multiplayer online role-playing games), 186
 MUD, 177, 186
 Myst, 185
 OXO, 174
 Pac-Man, 179, 180
 Pong, 178
 RTS (real-time strategy) games, 185
 RuneScape, 186
 Snake, 188
 Sonic the Hedgehog, 181
 Space Invaders, 178, 180
 Spacewar! game, 146, 174–175, 191
 Star Trek, 175, 176
 Super Mario Bros., 180–181
 Tetris, 188–189
 The Sims, 185
 World of Warcraft, 186
 Zork, 176–177
- garbage collection, 53, 64
 GCD (greatest common divisor), 85, 85
 GNU, 45, 72, 73, 245
 go to statement, 61, 62–63
 graph problems, 90–96
 minimal spanning tree, 92, 93
 shortest path algorithm, 93–94
 seven bridges problem (Bridges of Königsberg problem), 90–92, 99, 315–316
 traveling salesman problem, 94–96, 97, 98, 99, 100–101
- graphical processing units (GPUs), 186
 GUI (graphical user interface), 156–158, 162
- “half-adders,” 29, 30
 Hamiltonian paths, 99–100
 Hamming codes, 206
 hard disk, 37–38, 54, 166
 hardware, 23–38
 Harvard
 Mark I computer, 4, 5, 9, 13, 21, 22, 47–48
 Mark II computer, 9, 48
 Heath Robinson machine, 18
 heuristics, 247, 266
 Hidden Markov Models (HMM), 292
 hierarchical, 1–2, 23, 29–30
 high-level languages, 49, 59, 63
 Homebrew Computer Club, 147–149
 Hopfield networks, 327
 HTML (HyperText Markup Language), 226, 227, 233, 239, 241
 HTTP (HyperText Transfer Protocol), 226–227, 259
 human body tracking, 288–290, 291
 Human Genome Project, 315
 humanoid robots, 321
 hypercube networks, 140
 hypertext links, 221–223
 hypertext system, 162, 221–223
 hypertext transfer protocol (HTTP), 226–227, 259

IANA (Internet Assigned Numbers Authority), 212
IAS (Institute for Advanced Study), 17, 10, 37
IBM, 37–38, 49, 50, 52–53, 57, 58, 74, 133, 151–154, 156, 159, 164, 171, 271, 273, 294–296, 299–300, 341
ICANN (Internet Corporation for Assigned Names and Numbers), 229, 230
imperative programming languages, 60, 64
IMPs (interface message processors), 203–204, 205–208, 211
instruction, 39–40, 43–45, 166
integrated circuits, 124–128
Intel, 129, 130, 131, 133, 135, 144, 146–147, 152, 178, 210–211, 301, 302
Internet, 193, 211–213, 224, 229, 230, 232, 243, 259, 323–324
interrupts, 52, 53, 54, 55
ion-trap systems, 312, 312–313
IP (Internet Protocol), 79, 205, 211–213, 225, 229, 243
iPad, 155
iPhone, 155, 160, 340–341
iPod, 155
ISO (International Organization for Standardization), 212, 241
J. Craig Venter Institute, 315
Java programming language, 67
JavaScript, 76
job scheduling, 53–54
joystick controls, 178
kernels, 78
key logging, 250
Kinect, 182, 288–290, 291
lambda calculus, 111
laser printers, 143, 167
LEO (Lyons Electronic Office), xi, 9, 20, 88
libraries, 224
LIFO (Last In, First Out) stack, 46–47, 65
linked lists, 64–65
Linux operating system, 73–75
LISP programming language, 53, 60, 63–64, 111, 267
Loebner Prize, 266
logic gates, 27–30, 316
loop, 47, 61, 62, 63, 263–264
Lorenz machine, 18, 19, 253
MAC project, 53, 78, 161
machine code, 45, 46
machine learning
 frequentist approach, 280, 281, 286
 Jeopardy! (game show), 293–296, 297, 341
 mind-body problem, 325–327
 speech and language processing, 291–292, 293
 statistical language translation, 292, 293
 strong AI versus weak AI, 296–297
 universal grammar, 291
Macintosh computer, 155–157, 173

magnetic, 32–33, 36–37, 219
malware, 244
Manchester “Baby” computer, 10, 22, 35
Manchester Mark I computer, 35, 37, 105, 271
Manhattan project, 4–5, 86–87, 88
Markov chains, 286
mathematics
 incompleteness, 104
 inconsistency, 104
 integers, 60, 107–109
 irrational numbers, 108
 natural numbers, 107, 108–110
 modular arithmetic, 255–256
 rational numbers, 108
 real numbers, 60, 108–110
MATH-MATIC language, 49
MeCam quadcopter, 323
memex, 162, 166, 220–223
memory, 30–33, 36–38
 “cache memory,” 32
 cathode ray tube, 35, 36
 hard disk technology, 37–38
 hierarchy of, 31–32
 magnetic core technology, 36–37
 magnetic drum technology, 37
 “main,” 31–32
 mercury delay line, 35, 36
 random access, 33, 37
 “secondary,” 32–33
memristors, 304–305
MESM class computers, 21–22
message, 140, 198–200, 202–203, 212
microcode approach, 57
microprocessor, 130, 131, 133, 135, 144, 146–147, 152, 178, 183
micro-programming concept, 9, 57
Microsoft, 70, 71–72, 145–147, 148, 152, 153, 154, 157–158, 230–232, 237, 239, 244, 250–251, 259, 292
minicomputers, 164–166
Minix operating system, 73
Minuteman missile system, 128, 129
MIT (Massachusetts Institute of Technology)
 Lincoln Laboratory, 201–202
 Memory Test Computer, 37, 164
 networking experiment, 201–202
 time sharing, 53
 Tinker Toy computer, 30
model checking, 79
Mona Lisa problem, 96, 97
Monte Carlo methods, 86–88, 286
Moore School of Electrical Engineering, 3, 5, 7, 9–10, 35–36
Moore’s Law, 131–133, 301
 Dennard scaling, 130, 132–133
 fabrication facilities and silicon foundries, 133–134
 geometric and equivalent scaling, 300
 possible end to, 134–135, 302
Morris worm, 248–249
Mosaic browser, 224, 228, 229
MOS Technology, 149
mouse, 156, 162
Mozilla Foundation, 232
MS-DOS (PC-DOS), 152, 153, 154, 158
multicore architecture, 134–135
Multics (Multiplexed Information and Computing Service) operating system, 53, 78, 161
MYCIN program, 269
MyLifeBits project, 166
nanotechnology, 298–300, 301, 347–349
 carbon allotropes, 302–304, 305
 miniature motors, 298, 299
 miniature writing and engraving, 299–300
NASA, 82, 83, 129, 321, 322
Netscape Communications Corporation, 228–229, 230, 232, 259
networks
 bandwidth, 199
 broadband, 213–214
 centralized, 198
 circuit-switching, 199
 decentralized, 198
 distributed, 198–200, 202–203
 local area (LAN), 210
 message-switching, 198–200, 202–203
 packets, 197, 200, 202–203
 packet-switching, 197, 198, 199–201, 202–203
 store-and-forward, 196–197, 200, 202–203
 wide area (WAN), 192
neural networks, 273–277, 327–330
 artificial neurons, 275–277
 auto-associative memories, 327
 back-propagation networks, 276–277, 327
 deep learning approach, 277
 Hopfield networks, 327
 memory-prediction framework, 327–328, 330
 perceptron model, 275–276
 structure of the brain, 273–275
Newton message pad, 160
Nintendo, 180, 181, 182, 188–189
noncomputable numbers, 110, 112
NOR operations, 28, 29
NOT operations, 24, 28, 29, 307–308
NPL (National Physical Laboratory), 105, 200, 201
NSA (National Security Agency), 246, 248–249, 254
NSFNET, 213
numerical algorithms, 86–88
object-oriented (O-O) programming, 58–59, 60, 65–67
 classes, 65, 66
 encapsulation, 66
 information hiding, 65–66
 inheritance, 66, 67
 instances, 66
 methods, 58–59, 66

- object-oriented (O-O) programming (*cont.*)
 objects, 58–59, 66
 origin of term, 66, 168
- Observer pattern, 81
- ontology, 269–270
- open source software, 59, 72–75
- operating systems, 51–52, 53–56
 batch processing, 51–52
 device drivers, 53, 54
 file management, 54
 hardware interrupts, 54
 interrupts, 53
 job scheduling, 53–54
 layers of, 53
 origin of, 51–52
 security, 55–56
 system calls, 54
 time sharing, 52–53
 virtual memory, 55
- OR operations, 24, 25, 27–28, 29
- Osborne-1 portable computer, 170
- page faults, 55
- PageRank algorithm, 233–236
- parallel computing, 129, 134–135, 139, 140
- parity, 107, 115–116, 206, 312–313
- PC-DOS (MS-DOS), 152, 153, 154, 158
- PDP, 145, 146, 158, 164, 165–166, 174–175, 191, 205
- Perl (Practical Extraction and Report Language), 76
- personal digital assistants (PDAs), 159
- PGP (Pretty Good Privacy), 257–259
- Pilot ACE (Automatic Computing Engine), 8–9, 105, 200
- pixels, 183
- Plankalkül programming language, 21
- pointers, 65
- polynomial time algorithms, 97–100
- Pony Express, 195
- posteriors (posterior beliefs), 281
- Principia Mathematica, 103, 266–267
- processor-memory-switch (PMS)
 notation, 166
- processors, 33–34, 44
- program counters, 43, 46
- programmable logic controllers (PLCs), 252
- programming languages, 59–63,
 65–67, 75–76
- protocols, 206–207
- pseudocode, 61–62
- pseudo-random numbers, 88
- punched cards, 14, 15, 32, 39, 51
- Python programming language, 76
- quantum charge coupled devices (QCCDs), 312, 313
- quantum computing, 302, 305–313, 349–351
 error-correction techniques, 312–313
 ion-trap systems, 312, 312–313
 Josephson junctions, 307, 313
 key elements of, 306–307
 quantum entanglement, 308–310
- quantum factorization, 306, 311
- quantum parallelism, 308
- quantum search, 311
- reversible computers, 307–308
- topological, 313
- quantum theory, 131–132, 137–138
- quantum tunneling, 131–132
- Quantum Universal Turing machines, 305
- qubits, 306
- queuing theory, 203
- Quick and Dirty Operating System (QDOS), 153, 173
- quintuples, 110, 115
- RAM (random access memory), 33, 37, 129, 130, 133
- RAND, 95, 197
- real-time strategy (RTS) games, 185
- rectification, 123
- recursion, 59, 63–64
- Red Dwarf* (TV series), 344–345, 346
- Red Hat, 74
- redundancy, 198
- relational database model, 80
- relay, 24–25, 30
- remote log-in applications, 207
- rendering, 183–184
- Reverse Turing Test, 266
- reversible computers, 307–308
- RFCs (requests for comments), 206, 212
- RISC (reduced instruction set computing)
 architecture, 171
- Robot World Cup (RoboCup), 321
- robots, 320–323
- Rock’s Law, 133
- ROM (read-only memory), 129
- rootkits, 247–248
- RSA encryption, 256–257, 311
- R.U.R. (play), 320, 341, 342
- SAGE (Semi-Automated Ground Environment) air-defense system, 141, 161, 182, 183, 197
- sandbox games, 187, 347–349
- Sandstorm driverless vehicle, 322
- science fiction, 333–336, 347–349, 352
 alternate worlds, 351–352
 cyberpunk, 355
 English counterculture, 344–347
 magazines, 336–337
 nature of reality, 343–344, 345, 346
 robotics, 338, 341–343, 344
 scripting languages, 75–76
- search engines, 233–234, 235–237
 AltaVista, 167, 233–234, 235
 BackRub, 235
 Bing, 237
 Google, 187–188, 234, 235–236, 237, 239
 search engine optimization, 236–237
- search technology, 232–237
- security issues, 55–56
 buffer overflow, 249
 cookies, 259–260
 cyberespionage, 245–246
- cyberwarfare, 252
- denial of service attacks, 250
- hackers, 55–56, 209, 244, 353–355
- key logging, 250
- malware, 244
- rootkits, 247–248
- spam, 243–244, 261
- spoofing, 243
- spyware, 260
- Trojans, 246
- viruses, 246–247
- worms, 248–249, 250
- Semantic Web, 239, 269–270
- semaphores, 194–195
- Semiconductor Roadmap, 133
- semiconductors, 120–121
 doping, 122–123
 n-type, 122–123
 p-n junction diodes, 123
 p-type, 123
- SGML (Standard Generalized Markup Language), 241
- shells, 78
- Shockley Semiconductor Laboratory, 126
- SHRDLU program, 268, 291–292
- Silicon Valley, 121, 126, 128–129
- SIMULA 67 programming language, 65, 67
- Sketchpad program, 182
- SLAC (Stanford Linear Accelerator Center), 227, 228
- Slammer worm, 249
- Sloan Digital Survey, 80
- Smalltalk language, 66, 168
- smart phones, 159–160
- SMTP (Simple Mail Transfer Protocol), 243
- software, 39–56
 branches, 47
 complexity of programming, 40–41
 file clerk model, 41–44
 instruction sets, 39–40, 43–44
 loops, 47
 machine code, 45, 46
 microcode, 57
 subroutines, 46–47, 63
- software engineering, 6, 58–76
 agile software development, 68–69
 dynamic data structures, 59, 64–65
 linked lists, 64–65
- empirical, 70–72
- for loops, 62, 63
- formal methods, 79
- if-then-else constructs, 62, 63
- recursion, 59, 63–64
- software crisis, 58–59
- software errors, 79, 82, 83
- software life cycle, 69–70
 debugging, 59
 requirements analysis and specification, 69
 testing and maintenance, 59, 69–70
- strong typing, 61

- “waterfall” model of development, 68, 69
- solid state technology, 120–122
- insulators, 121, 122
 - integrated circuits, 124–125
 - CMOS technology, 127–128
 - fabrication facilities and foundries, 134
 - invention of, 125–126, 128
 - mass production of, 126–127
 - military market for, 128, 129
 - metals, 121–122
- source code, 59
- SourceForge, 75
- sparse matrices, 235
- Spirit rover, 321
- sprites, 183
- Sputnik satellites, 197
- spyware, 260
- SQL (structured query language), 80
- SRAM (static random access memory), 129
- SSL (secure sockets layer), 228–229, 259
- Stanley driverless vehicle, 322
- Star Trek (TV and film franchise), 339, 340
- Star Wars (film series), 342
- statecharts, 69, 70
- STM (scanning tunneling microscope), 299–300
- stored-program computers, 3, 5–7, 9, 10, 35, 45
- structured programming, 58, 63, 79
- Stuxnet worm, 252
- supercomputers, 139–140
- switching circuits, 24–25
- synthetic biology, 313–316
- system administrators, 244, 245
- system calls, 54
- tags, 226, 233, 238, 239, 241
- TCP/IP (Transmission Control Protocol/Internet Protocol), 79, 205, 211–213, 225
- telegraph, 193, 194–197, 219, 336
- Telnet, 207
- Texas Instruments, 125, 129
- third-party cookies, 259–260
- thought experiments, 281–282, 296–297
- time sharing, 52–53, 78, 142, 164, 202
- Tinker Toy computer, 31
- total internal reflection, 214
- Total Turing Test, 326
- touch screen input, 155, 159–160
- transatlantic cable, 196, 197
- transistors, 124
 - early uses for, 124–125
 - FETs (field effect transistors), 301–302
 - FinFET transistors, 301–302
 - mesa, 126, 127
 - MOSFETs (metal-oxide-semiconductor field effect transistors), 127–128
 - origin of term, 124
 - planar, 126, 127
 - p-n-p junction, 124
 - point-contact, 123–124
 - 3D transistors, 301–302- tree structure organization, 92, 93
- truth tables, 27–28, 29, 30, 32, 308
- Turing machines, 18, 105–107, 114–116
 - computationalism, 325
 - halting problem, 112
 - parity counter, 107, 115–116
 - universal, 2, 110–111
- Turing Test, 105, 264–266, 325, 326–327
- Twitter, 239, 319
- Unified Modeling Language (UML), 69, 70
- UNIVAC computer, 3, 9–10, 48–49, 337–338
- universal resource identifiers (URIs), 226
- universal translators, 339
- universality, 2, 7
- Unix operating system, 72–73, 75, 78–79, 212–213, 225, 244
- variables, 60
- VAX-11 computer, 165–166

virtual memory, 55, 166

virtual reality, 352–353

viruses, 113, 246–247

VisiCalc application, 150–151, 155

VMS operating system, 166

W3C (World Wide Web Consortium), 228

Web 2.0, 237–239

web browsers, 224, 227, 228, 229, 230, 232, 259

web crawlers, 233

web servers, 226–227

web spam, 236–237

Wheeler jump, 46

Wikipedia, 238, 269–270, 319

wikis, 237–238

WIMP (windows, icons, menus, and pointers) interface, 156

Windows operating system, 156–157, 232, 287

Winsock (Windows sockets interface), 230–231

Word program, 157

workstations, 193

World Brain concept, 334, 335

World Wide Web, 224–226, 227, 228–230, 232–237, 319

WYSIWYG concept, 157, 167, 168–169

Xanadu project, 222–223

Xbox, 181–182, 186–187

Xerox, 72, 154–155, 210–211

Xerox PARC (Palo Alto Research Center), 66, 133, 142–143, 154–155, 156, 163, 191, 202, 209–210, 248

Yahoo!, 232, 233, 236, 239

Yandex, 239

Z80 microprocessor, 170

zero day bugs, 252

zombie computers, 244, 250