NATURAL COMPUTING

NATURAL COMPUTING

DNA, Quantum Bits, and the Future of Smart Machines

DENNIS SHASHA
and CATHY LAZERE

W. W. NORTON & COMPANY
NEW YORK LONDON

Printed in the United States of America
First Edition

Manufacturing by Courier Westford
Book design by Lovedog Studio
Production manager: Devon Zahn

Library of Congress Cataloging-in-Publication Data

Shasha, Dennis Elliott.
Natural computing : DNA, quantum bits, and the future
of smart machines / Dennis Shasha and Cathy Lazere. — 1st ed.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-393-33683-2 (pbk.)
1. Natural computation. 2. Artificial intelligence.
3. Computer scientists. I. Lazere, Cathy A. II. Title.
QA76.9.N37S33 2010
006.3—dc22
2010004807

W. W. Norton & Company, Inc.
500 Fifth Avenue, New York, N.Y. 10110
www.wwnorton.com

W. W. Norton & Company Ltd.
Castle House, 75/76 Wells Street, London W1T 3QT

1 2 3 4 5 6 7 8 9 0

CONTENTS

PREFACE

When we began this book, we expected to talk to scientists who were solving problems on the frontier of what is possible in computing. We expected them to describe new computer architectures and a variety of new software techniques. The fifteen scientists featured in *Natural Computing* control spacecraft from millions of miles away, embed intelligence in smart bacteria, or build computers to run as fast as a million desktops combined. They work on the most challenging applications in science, engineering, and even finance. We expected this diversity, but we didn't expect the common vision that has emerged across all of these fields: *the future of computing is a synthesis with nature.*

There are three major strands to this vision.

First, biological thinking has inspired new ways to do digital computing. This hasn't happened yet in your word processor or in data centers, but it has occurred in applications that are pushing technology to its most interesting limits. For example, computers control spacecraft throughout flight and after landing. Manual repair of onboard hardware, once in flight, is often impossible;

but innovative spacecraft engineers propose designing machines that will repair themselves. If you're not in an engineering field, you may not appreciate what a change in thinking this approach represents. Instead of building a high-precision machine that can handle every possibility, designers construct a machine that will adapt itself to possibilities the designers cannot even imagine.

Although there is some debate about how to realize this new design philosophy, many of the scientists profiled in this book suggest that evolution or some form of learning should be involved. At first glance, evolution and learning may seem very different. Evolution works across many generations and organisms, while learning and adaptation occur in a single organism. Conceptually, however, they are very similar: try something and see how it works; then respond to feedback by trying something new that has a chance to be better. We associate learning with conscious improvement, but evolution occurs subconsciously, chemically, or even metabolically. For purposes of survival, however, learning and evolution have the same effect.

Many applications are based on evolutionary techniques. In *Natural Computing*, a surfer dude turned mathematical financier uses evolutionary algorithms to figure out when to buy or sell US Treasury bonds. A professor analyzes the safety of a missile defense system by checking how well safety procedures adapt to failure. In these instances, evolution, learning, and adaptation are linked.

Second, biological entities may replace silicon. Computing built on DNA or bacterial cells is nearly free (billions of bacteria need just a little sugar water to grow), and DNA computing also has a massively parallel capacity. One day, living bacteria rather than silicon electronics may compute inside our bodies or inside

microfactories. Harnessing life in this way will require the development of an entirely new paradigm of computing. No longer will a human hand design computers to work reliably for several years, give them universally known names, and install them in air-conditioned splendor.

Computers made of bacteria or viruses come by the million, have no names, and are provincial—they communicate only with their neighbors. They also fail often. Getting them to do something useful is daunting and might seem impossible, but it must be possible because we already have what mathematicians call an existence proof: humans are composed of about 100 trillion cells and we are able to run, think, and love, even though none of our individual cells can do those things.

Third, new applications may require rethinking the underlying physics of computation. Simulating protein folding or breaking codes requires thousands of processors to coordinate their work. Because communication through switches and computer memory is slow compared with the speed of transistors, such a strategy works best when there is little communication between faraway processors. For that reason, the fastest digital machines on the planet are designed to move information as little as possible.

For example, a multi-millionaire inventor has embarked on the design of a machine to simulate the behavior of proteins. To do this, his hardware moves the information concerning each atom in the protein to very specific places within the active circuitry. At those places, information is combined with corresponding information about neighboring atoms—all without returning to the computer's slow memory. The payoff may be speeding up computation by a factor of 1,000, enough to accel-

erate a task that would normally require as much as a century to one that could be completed in less than 100 days.

Another designer, much less well funded, has constructed a computing device whose main computational element is a piece of foam attached to 25 wires. Measuring the electric current could offer a method to simulate the differential equations that characterize a whole host of "continuous" scientific problems, from the prediction of star trajectories in galaxies to the propagation of pigmentation on butterfly wings. This "extended analog computer" turns computing completely on its head. Instead of *calculating* an answer using 1s and 0s and arithmetic in a digital computer, it *measures* an answer.

You may find yourself feeling affection for these machines. Instead of the hard metal engines that do calculations or send e-mails or play chess, you will see machines that are more human—they repair themselves, attempt extremely hard problems, and sometimes make mistakes. These are computational devices that may one day set your broken bones, maintain the stability of bridges, or perhaps even help you breathe underwater.

In writing *Natural Computing*, we encountered a constellation of ideas that could change the world decisively for the better. Each chapter in this book describes a unique path to discovery. We hope you enjoy reading the stories of these risk-seeking adventurers as much as we enjoyed writing them.

Part I

ADAPTIVE COMPUTING

MODERN MANUFACTURING BEGAN WITH THE NOTION OF INTERchangeable parts, dating back to Johannes Gutenberg's movable type in the fifteenth century. By the eighteenth century, manufacturers had become more concerned about the precision of the parts. Eli Whitney's interchangeable musket parts had a precision of 1/30 inch (about 1 millimeter). Machine tolerances today are typically 10 micrometers (millionths of a meter), 100 times more precise than what Whitney could achieve. Optical tolerances are now measured in the nanometer range—a million times more precise than Whitney's. Designers now have the opportunity to build machines of exquisite precision for the task at hand.

Mainstream computer science is built on algorithms. An *algorithm* is a method that is guaranteed to produce a correct response for a large class of stimuli with a specified efficiency. Think of algorithms as recipes—to produce a particular dish, combine specified ingredients in a recommended order to obtain

a desired result. For example, a "mergesort" algorithm puts items in order no matter what kinds of items are presented to it.

Although algorithms will always play a central role in computing, some problems are fundamentally not algorithmic. Consider the following problem: You are to survive in Antarctica and keep equipment operating at any temperature down to –60°C (–76°F). You know that your shelter and clothes may suffer any one of many possible mishaps. How will you and your equipment survive? An algorithmically oriented computer scientist would complain that the problem is ill posed. If the mishaps are great enough, it may not be possible to survive. But what if you had to design a solution that would work most of the time? You would need to build in adaptation or its cousin, evolution.

As early as 1954, the mathematician Nils Aall Barricelli, working at the Institute for Advanced Study at Princeton, simulated simple models of evolution using a computer. About fifteen years later, the German computer scientists Ingo Rechenberg and Hans-Paul Schwefel used evolution to improve designs for complex engineering problems.

In the natural world, evolution applies to organisms. In the computational world, evolution applies to designs. In both cases, evolution can lead to beautiful results *without the benefit of a conscious designer.* In 1975, John Holland of the University of Michigan wrote a landmark book, *Adaptation in Natural and Artificial Systems*, in which he showed the commonalities among the different approaches to evolutionary design and improved them through a uniform mathematical framework.

Holland's framework became the basis for modern genetic (sometimes called "evolutionary") algorithms. It consists of repeated applications of the following procedure:

1. Start with a population of possible designs (candidates).
2. Evaluate each one to give a "fitness" score, perhaps based on monetary cost or energy consumption.
3. Remember the design that receives the best fitness score.
4. Create a new population by selecting the fittest candidate designs and changing them slightly in a random way or changing them greatly by combining different designs together.

Suppose you are using this method to design a car. If a good design proposes a composite chassis with a six-cylinder engine and another good design proposes an aluminum chassis with an electric engine, the combined design might be a composite chassis with an electric engine. Parent designs beget children having some characteristics of each.

Although evolution can lead to better designs, small adaptations require less effort. For example, you can learn to ride a bicycle or juggle without evolving. Adaptations at that level may entail trial and error, but the organism doesn't need to change. **Rodney Brooks** does adaptation in motion. Since the 1980s, he has designed robots that move intelligently by adaptation. Since starting his pioneering work, Brooks has been inspired by insects, elephants, and geckos. In the process, he has redefined what it means for robots to be smart.

Suppose you are designing software for a robot that must navigate the surface of another planet. You don't know what the exact task will be. You do know the ground is rough. You also know the environment is extremely hostile, but you don't know the particulars. You are faced, in fact, not only with unknowns but with what **Glenn Reeves** of NASA's Jet Propulsion Labora-

tory calls "unknown unknowns"—unknowns you can't even characterize. It won't work to design a rover, send it, and then hope for the best. In fact, the current state of the art is to diagnose from afar—100 million miles away. To do that, Reeves has to design spacecraft instruments that act like chatty patients talking about their current condition and reporting when they feel better. When the instrument is sick, his team then has to send up electronic prosthetics in the form of *patches*, or small changes to the software. The device uses the patch in place of the erroneous code in the same way a patient would use a prosthetic limb in place of a non-functioning limb.

Adrian Stoica, who also works at the Jet Propulsion Laboratory, imagines future spacecraft that can heal themselves. Consider the challenges: day and nighttime temperatures on the surface of, say, Mars might vary from a frigid –133°C (–207°F) to a balmy +27°C (+80°F). A person facing temperature variations changes clothes. A circuit can't cover itself, but perhaps it can change how the electrons flow. Stoica has designed circuits that can "evolve" a solution on their own. Stoica dreams of equipment that will survive 100 years by evolutionary adaptation.

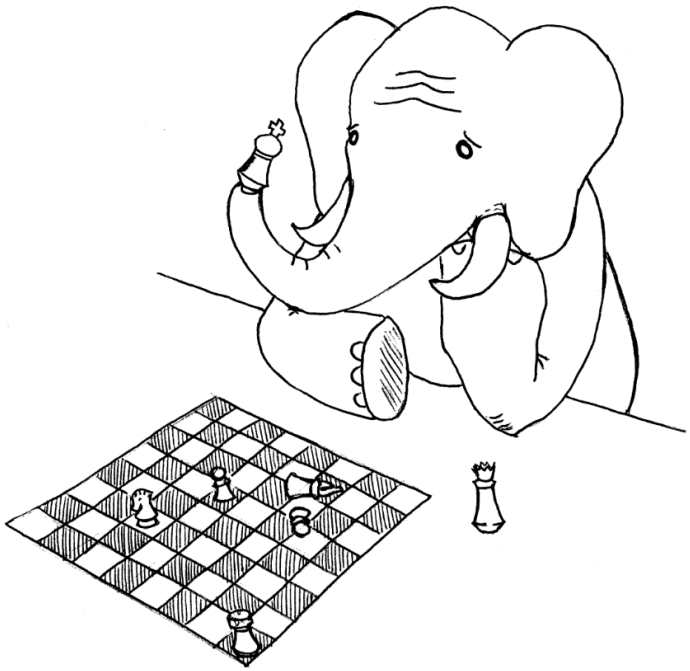
Some would argue that genetic algorithms do not deserve the good name of *algorithm*, because they don't guarantee efficiency and correctness. This is true. A genetic algorithm offers no guarantees, but it often arrives at surprisingly good designs for problems having no known algorithms. Using genetic algorithms, **Louis Qualls** designs custom nuclear power plants for extreme environments, including space. To get the specifications—how much power the plant should generate, how much it should weigh if powering a spacecraft, and so on—he talks to specialists, gets design preferences from each of them, and then tries

to arrive at a design that embodies a compromise among those preferences while meeting the specifications. To do this, he must choose among trillions of design possibilities. By hand, he can explore only a fraction of the possibilities in his search for a good design. He also knows that he often returns to designs he has done previously because he has grown emotionally attached to them. By contrast, when he uses genetic algorithms to program his computer, the programs sometimes discover designs that he would never have considered but either cost less or perform better than his own. Further, if the specifications change (and they often do), he can set the computer off to find another design, still without any emotional attachment. It is quite likely that in the future, most complex engineering artifacts will be designed in this way.

Jake Loveless and **Amrut Bharambe** apply similar ideas to finance and use genetic algorithms to design rules to help them trade treasury bonds. While engineers use complex physical criteria to determine "fitness," Loveless and Bharambe use the most basic financial measures: high profit at relatively low risk. To determine whether a rule is good, they try it on historical data. The *search space* (the number of possible rules) is scarily large and they don't understand the rules that their genetic algorithms generate, but the method works.

Nancy Leveson works with nature too, but mostly human nature as it meshes with high technology. She considers the marvels of engineering—power plants, missile defense, and spacecraft—and tries to make them safe. She started her career in computing, but then took off for the jungles of New Guinea and pursued a serious interest in cognitive psychology before returning with a new perspective on computing. She believes that safety requires

dependable adaptation—mistakes happen, but the system must compensate for them. The "system," in Leveson's view, does not stop at a convenient boundary like software specification. It extends all the way up the human management chain. When computing elements combine with people in life-critical situations, adaptability translates to multiple levels of feedback and modification. A failure at a lower level must be compensated for by a person or machine at a higher level. Leveson's approach tries to ensure that every level of a system detects problems at the level below and responds appropriately. Conceptually simple, this approach is being used to help prevent accidental missile launches and air traffic accidents.



You're all assuming there's a logical representation inside the robot. What if there is no logical representation? I had been watching insects and how they do stuff . . . do they really have a three-dimensional rendering of the world around them—a computer graphics model inside that puny little head with 50,000 neurons? Is that what those neurons are doing?

— *Rodney Brooks*

Chapter 1

RODNEY BROOKS

. . . .

Animals Rule

WHEN ARTIFICIAL INTELLIGENCE (AI) WAS BORN IN THE 1950s, excelling at IQ tests or chess seemed to be a good indication of intelligence. After all, that's what schools measured. Since then, a slew of other definitions have been added to the mix, including emotional IQ and Howard Gardner's interpersonal and kinesthetic measures of intelligence. But if we define intelligence as the ability to survive in the world, we need to look at more fundamental skills. How is it that we can walk, recognize objects, and navigate around obstacles? You may say, "Animals can do that!" To which Rodney Brooks might respond, "Exactly!" In fact, robots might do better if they didn't copy humans in all aspects of our behavior. For example, who walks better over rough terrain—humans or insects? If you've ever seen insects scramble out of impossible holes, you might vote for the insects.

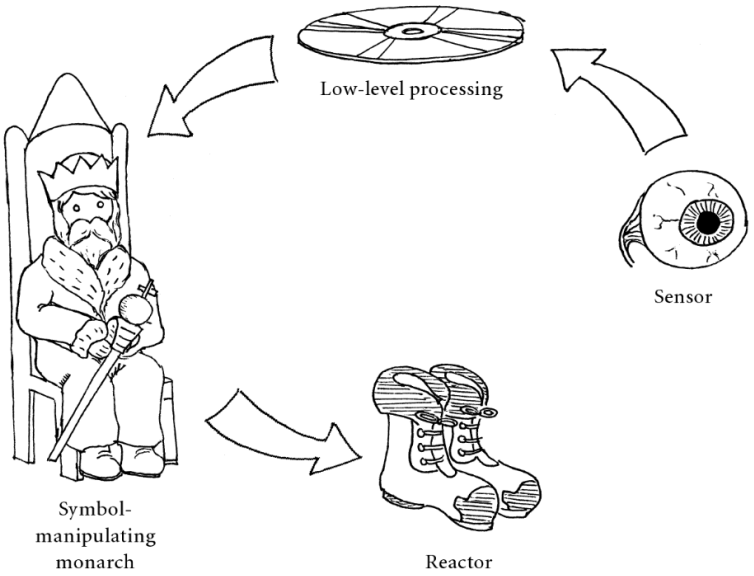
In a seminal paper from 1990, "Elephants Don't Play Chess," Rodney Brooks presented an evolutionary argument for the relative insignificance of human "higher" intelligence. Life arose

on Earth 3.5 billion years ago, he noted; vertebrates and insects appeared during the last 10% of that time, approximately 450 million years ago. The great apes emerged in the last 0.5% of that period, about 18 million years ago. Agriculture was created only 19,000 years ago, 0.0005% of life's time on Earth. Expert knowledge has appeared only in the last few hundred years.

Computers are most successful at rapidly performing the skills learned in the last few hundred years of human history, perhaps because we are most conscious of those skills and they take the most conscious effort. But our unconscious acts pose a greater computing challenge. In the brief history of space travel, it has been easier to build a computer program to guide a spacecraft to Mars than to build a robot able to navigate over rough terrain with anything like the skill of a billy goat. Evolution required billions of years to arrive at the billy goat, but only a few million more to arrive at human intelligence. The factor of 1,000 in relative timescales should give us a certain humility before these "primitive" intelligences.

When Brooks wrote his "Elephants" paper, the field of artificial intelligence modeled intelligence as symbol manipulation. The scientific goal was to design sensor modules such as vision systems. These would abstract the world into symbols and pass the symbols into an intelligent core, a kind of electronic monarch. The monarch would manipulate the symbols and then instruct actuators (normally wheels) to move. In many ways, this stepped process mirrored the idealized hierarchy of a large corporation or the military—"brains" on top, eyes and limbs on the bottom. Brooks objected to this paradigm on philosophical as well as pragmatic grounds.

Brooks's philosophical objections may have originated in his



The monarch model. The monarch receives data from sensors, reasons about it, and then emits decisions that are carried out by reactors.

unlikely path to science and MIT, where he is now a professor at CSAIL, the Computer Science and Artificial Intelligence Laboratory. Brooks was born in 1954 in Adelaide, Australia. Though not exactly the outback, Adelaide was a long way from the centers of computer science research, but the remoteness may have been to Brooks's advantage. Nobody told him the right way to approach the field. At age eight, on his own, he began designing computers to play games. At twelve, he built a primitive computer out of old telephone relays to play tic-tac-toe. He resolved to pursue a career in game design.

In 1972, Brooks began studies at Flinders University of South

Australia. On the weekends he was permitted to use the lone university computer with its 16 kilobytes (roughly 16,000 bytes) of memory and a 1-megabyte disk. Its million bytes were only one-millionth of the memory capacity of a contemporary desktop computer. Still, a megabyte was a lot more than 16 kilobytes. Brooks figured out how to program the computer as if it had the full 1 megabyte of memory by moving data from the disk when necessary. He used an innovation called *virtual memory* that had been commercially realized only a few years earlier. Brooks didn't look for papers describing how to do it; he just did it. "Someone had described to me the idea," says Brooks. "It sounded pretty good, so I implemented virtual memory on this computer."

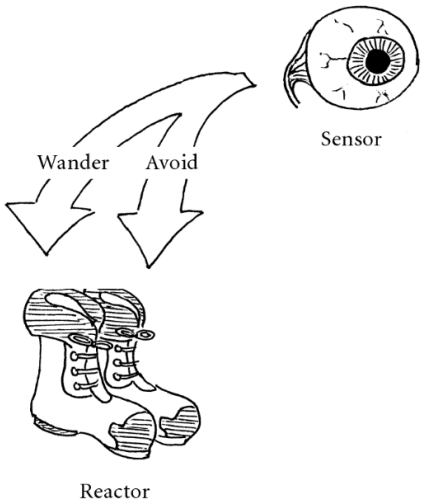
Brooks went to Stanford for his PhD, where he met another up-and-coming roboticist: Hans Moravec, a graduate student a few years ahead of him. In the summer of 1979, in a little lab in the hills behind the campus, Brooks helped Moravec with a robot known as Cart. At midnight each evening, when everyone else went home, Brooks and Moravec would set up Cart. The robot would move about 20 meters around the lab for the next 6 hours. The two young scientists wanted to create *stereo vision*—giving the robot depth perception by equipping it with slightly different images from each of two viewpoints. Stereo vision normally requires two cameras, but at that time cameras were expensive and they had only one. So they had to slide the one camera from side to side. "I was just a gofer to move furniture around, set things up, get things connected," says Brooks. The computer would have a look at the world and then compute for 15 minutes and move a meter. Then it would open its eyes, look, shut its eyes, and compute for another 15 minutes. It would move a meter blind, based

on where it thought things were. "At the time I thought it was way too much computation," notes Brooks.

Brooks's doctoral thesis was on *machine vision*, a classic AI approach in which the camera would feed a computer an image. Brooks's program would then translate the visual scene into symbols to be processed by a hypothetical "intelligence"—the symbol-manipulating monarch. "The basic idea that nobody was questioning was that you've got a camera, you've got pixels, and you just change the pixels into a logical description of the world," Brooks says.

On vacation in Thailand in 1988, Brooks visited his first wife's family home, which stood on stilts by a river. No one spoke English, so Brooks sat by himself, watching insects. The more he watched, the more he began to question the symbolic AI paradigm. He just couldn't believe that insects were capable of forming logical descriptions inside "that puny little head with 50,000 neurons."

In 1990, Brooks's "Elephants" paper explained what he playfully called "nouvelle AI." His hypothesis was that an intelligent system had to have its representations grounded in the physical world. "The world is the best model of itself," as he put it. The world is up-to-date and contains all necessary details. This meant that Brooks's robots would dispense with the hierarchical structure of "classical" AI, with its symbolic representation of the world. Instead, nouvelle AI robots would possess a set of independently designed skills. Just as a human plays basketball and walks using the same limbs and eyes, a robot shares sensors and actuators for different skills. The skills, however, are independent—some of them, especially the highest-level ones, may fail without causing others to stop working.



Nouvelle AI. A robot should sense and then move according to simple rules such as "Avoid collisions" or "Wander."

Embodying this philosophy, the Brooks lab at MIT had built an early robot called Allen in 1985, which Brooks puckishly had named after Allen Newell, one of the early proponents of symbolic AI. Allen had three skills: avoid collisions, wander around randomly, and go to distant objects. "Allen would happily sit in the middle of a room until approached, and then scurry away, avoiding collisions as it went," remembers Brooks. "The internal representation was that every sonar return represented a repulsive force."

At the lowest level, Allen acted like a frightened mouse, following a primal rule: avoid hitting or being hit. What would keep Allen from hiding in a corner? Every 10 seconds, Allen would be told to wander randomly. Note that wandering also requires moving wheels, in accordance with the Brooks strategy of having

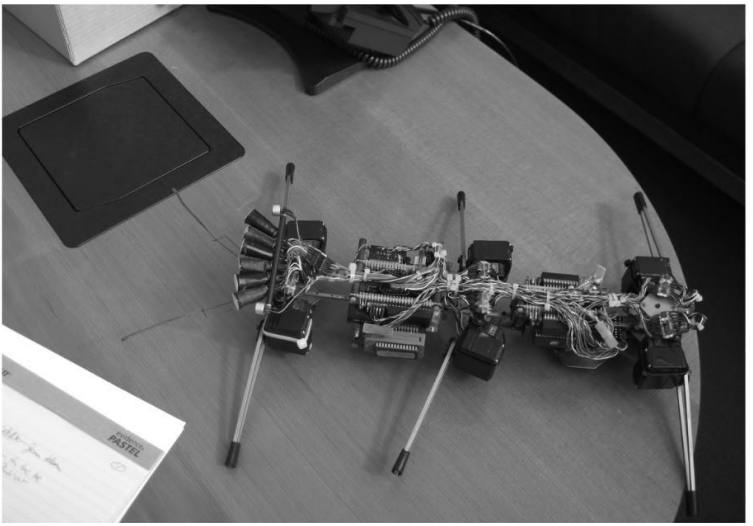
different skills use the same actuators. The collision avoidance skill takes precedence over wandering.

In its third function, the robot used its sonar to look for distant places and try to go to them. It would measure distance using an odometer. Like a runner trying to complete a mountain race while avoiding a slip off the edge, the robot combined goal seeking with underlying survival skills. It seemed almost too easy to be research. "I argue for simplicity," says Brooks. "Get away from hairy equations." He contrasts that viewpoint with the belief of some of his colleagues and critics, who think the hairier the better. To Brooks, if you need to explain something with a lot of convoluted mathematics, the solution will be "pretty unstable." "I'm interested in building something that can't fail to work," he says.

Rolling robots were one thing. What about walking robots for scrambling over rough terrain? All that thinking in Thailand would be put into play. Working with Colin Angle and Grinnell Moore, a high school student, Brooks built a six-legged walking robot named Genghis. "There weren't many walking robots at the time. Everyone else's walking machine was big and fragile," remembers Brooks.

Brooks had been looking at high-speed videos of insects running. "They fall all the time and hit their metaphorical chin," he says. Because the strength-to-weight ratio is greater at smaller sizes, they can afford to be really bad walkers. By contrast, a filly at the Kentucky Derby can't afford to fall—she might break a leg and suffer a career-ending injury.

Brooks found a natural application for Genghis at the Jet Propulsion Laboratory (JPL). He started attending meetings as they were talking about new Mars missions. JPL was promoting a



Brooks's walking robot Genghis, a multi-legged, low-lying device that could afford to fall.

1,000-kilogram robot called Robbie with a big arm at the front. It moved a centimeter a minute. JPL estimated that a mission to send Robbie to Mars would cost about $12 billion. Brooks knew the required funding would never come through. At a meeting, he suggested sending a small robot instead of a big one. "I remember one of the lifers at JPL who was an instrument builder saying [as he tells the story, Brooks morphs his Aussie accent into a southern drawl], 'A scientist waits 15 or 20 years for a mission, he doesn't want a little bitty instrument—he wants a *big* instrument. You can't send a little robot—you've got to send a *big* robot!'"

Brooks proposed a compromise. Instead of sending a single 1,000-kilogram robot, send a hundred 1-kilogram robots. That would cut the total mass down by a factor of 10 and spread the

risk. "If you send a 1,000-kilogram robot, you have to be very careful about what you do," says Brooks. "If you screw up, you've lost your $12 billion investment. If you've got 100 of them and you lose one—big deal."

Brooks found another benefit in the "swarm of robots" idea: mass production. Building 100 copies of a small robot would be cheaper than building a single 1,000-kilogram robot. But then what do you do with 100 robots once they get to Mars? "If you have a single 1,000-kilogram robot, you want to control it at all times. If you have 100 [smaller robots], you can't possibly control them; they have to be autonomous. So get over it—make them autonomous from day one and let them out of your control."

In 1989, Brooks wrote a paper for the British Interplanetary Society embodying his in-your-face plan. The title of that paper—"Fast, Cheap, and Out of Control"—was later used in 1997 for a documentary by Erroll Morris in which Brooks appeared, along with a topiary artist, a lion tamer, and the world's foremost expert on naked moles. The paper resonated with a NASA scientist named Donna Shirley. Shirley arranged for Colin Angle, an undergraduate at the time, to spend the summer of 1989 at the Jet Propulsion Laboratory. There Angle built a little, half-kilogram robot called Tooth that could do many of the things the 1,000-kilogram JPL robot Robbie could do. Rajiv Desau and Dave Miller at JPL used the Tooth code to build Rocky I and Rocky II, six-wheeled rovers. The current Mars rovers are based on that design.

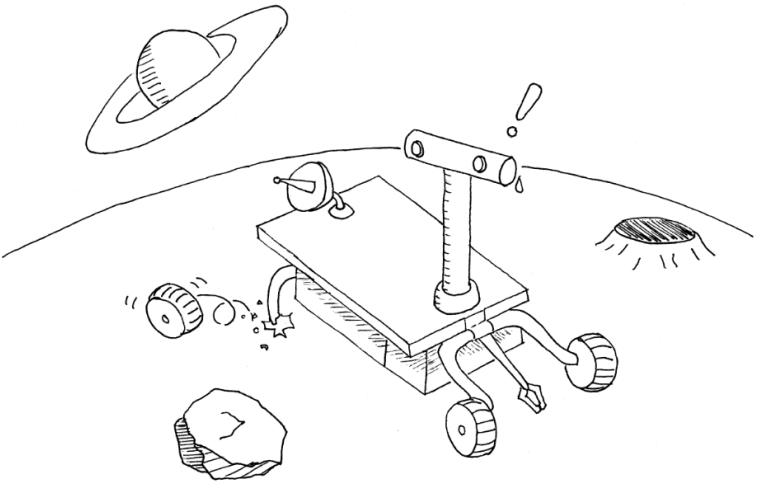
Inspired by that success, Brooks and the newly minted graduate Colin Angle decided to start a company for lunar and Mars exploration. "We were going to send pairs of robots to the moon

with advertising decals on them," says Brooks. Eventually, Brooks teamed up with David Scott, commander of *Apollo 15*. Through Scott, Brooks became connected in 1992 to the Strategic Defense Initiative (SDI), fondly labeled "Star Wars" by its opponents and, under Clinton, renamed the Ballistic Missile Defense Organization. The plan was to exploit bureaucratic rivalry. The developers of ballistic missile defense had spacecraft called Brilliant Pebbles. The idea was that these spacecraft would locate rocket plumes of ballistic missiles in their launch phase and then ram them with non-nuclear mini-missiles. But by 1992, the Soviet threat had disappeared, so the Ballistic Missile Defense Organization needed to show off its technology in other ways.

A new plan was proposed to put a Brilliant Pebble on a vehicle called Clementine that would orbit the moon. The missile detection instruments of the pebble would be replaced with Brooks's walking robot called Grendel. The pebble would land on the moon, and the robot would do some environmental tests. In 1993, a test run was conducted at Edwards Air Force Base. It looked as though it could work. Interagency turf wars quickly interfered, however. NASA protested that extraterrestrial exploration was its job, so the idea was scrapped. Brooks's Rocky VI, however, was sent to Mars. NASA seemed on board with the Brooks philosophy. "They said we're going to do it faster, cheaper, better. We're not going to do it the old way. We're going to do it smarter," Brooks recalls. Then the next mission to Mars failed. "Faster, cheaper, better got canned."

Since then, Brooks's company, iRobot, has been making autonomous vacuum cleaners called Roombas for household use. It also manufactures bomb-disarming robots for the military. The bioengineer Robert Full has collaborated with iRobot

on robot construction and has revisited the insect inspiration. After studying the movements of cockroaches and similar crawling insects with colleagues, Full determined that the legs of such creatures could be modeled as articulated pogo sticks. Designing moving robots this way enables them to navigate over difficult obstacles and to move in and out of craters without a vision system and with minimal computation. Springy limbs have eliminated the need for thought in locomotion. Simplicity wins.



■ ■ ■ ■

When you're working with equipment on Earth, how you diagnose it and how you repair it are based on the fact that quite literally you can walk up to it and touch it. The minute it leaves the Earth, none of those things exist. You can't touch it—you can only observe it by what it tells you. You can't go fix it. So you have to deal with the fact that if something is going to break, you either have to give up, or you have to work around it.

— *Glenn Reeves*

If you send out a spacecraft that will reach its target in 100 years, you can't afford to use electronics that will fail within 10 years. You can try massive redundancy, but then you might make a spacecraft that can't fly. The goal is to see if these systems can become long life survivable all by themselves.

— *Adrian Stoica*

Chapter 2

GLENN REEVES and ADRIAN STOICA

. . . .

Design for a Faraway Planet

IMAGINE YOU HAVE TO TROUBLESHOOT A MULTI-MILLION-DOLLAR production system at a customer site. All eyes are on you—the customer's, your boss's, and, if you're in charge, your colleagues'. To analyze the problem, you test individual software and hardware components, think, change the data, and run the software. They're watching you. You think some more, run the existing data on different hardware, think as clearly as possible. They're still watching. You think of a fix that might work. You try it . . .

Now imagine that your equipment is unique to your application. Not only are your customers looking at you, but so is the world press. The spacecraft is several minutes away at the speed of light. In the best case, weather and power permitting, it can communicate with you at 1,200 bits per second (a DSL line communicates 5,000 times faster). You can't change the hardware or even physically look at it.

Next, imagine that the spacecraft's mission is to a distant

planet, so far away that it takes 100 years to get there. Once there, the spacecraft is subject to extremes in heat and cold. If a part breaks, there's no FedEx to replace it. Now suppose that somehow the spacecraft can fix itself. Welcome to the challenges faced by Glenn Reeves and Adrian Stoica of the Jet Propulsion Laboratory.

Fix and Learn

Born in 1960, Glenn Reeves grew up in South Pasadena. His father worked as a civil engineer and manager for Caltrans, the California State department in charge of freeways. As a young boy, Reeves was interested in outer space. At his fifth birthday party, cardboard cutouts of the moon, the planets, and stars hung from the trees in the backyard. Other more grown-up dreamers lived in the neighborhood—Pasadena is the home of the Jet Propulsion Laboratory (JPL).

Set against the San Gabriel Mountains, JPL sprawls over almost two hundred acres. It was founded in the 1930s by the California Institute of Technology (Caltech). In the wake of *Sputnik*, JPL designed the first American satellite, *Explorer I*, which was launched in 1958, shortly before Reeves was born. JPL is essentially a laboratory for NASA; its employees design spacecraft and robotic missions to explore the moon and the planets.

"We would drive by JPL when I was young," remembers Reeves. "My mom would point as we were driving by and say to me, 'You know you're going to work there someday.' I didn't even know what the place was." In high school, Reeves pursued science, but only halfheartedly. He liked physics but didn't excel at it. He took a math self-study program and fell behind. His main passion was working as an auto mechanic. "I was far more inter-

ested in having a job and making some money than preparing for a career," he admits.

At high school graduation, Reeves knew he should go to college—his parents had insisted on that—but where to go and what to study escaped him. He planned to go to the University of California at Santa Barbara, but then he became romantically involved with a local woman and decided to attend Cal Poly at Pomona instead. There he took introductory courses in hotel and restaurant management, accounting, and information systems. It was his first exposure to the computers of the day, which required writing instructions on Fortran punch cards. "That little puzzle of how to do that very simple programming had me sufficiently intrigued to say, well, maybe I'd like to do this," Reeves recounts.

Reeves majored in computer science and continued living in South Pasadena, commuting every day to Pomona. A chance meeting changed his life. On his way to school one day, he noticed one of his professors, John Rohr, sitting at a bus stop. Reeves was en route to Rohr's class and realized that if Rohr kept waiting for the bus, the class wouldn't happen. So he offered Rohr a ride. In the course of their conversation during that drive, Rohr talked about his work on the software for deep-space missions at JPL.

Rohr became Reeves's advisor and arranged a summer job for Reeves at JPL. That short-term job led to a full-time position after Reeves graduated in 1983. He was assigned as a programmer for the spacecraft test equipment of the *Magellan* spacecraft. Named after the Portuguese explorer Ferdinand Magellan, the spacecraft's mission was to map the surface of Venus.

Spacecraft test equipment of that era had two roles. First, it had to act like the ground control equipment, the electronics that

communicates with the spacecraft. Second, it had to act like the spacecraft itself, including the parts that hadn't been built yet. With these functions, the test equipment enabled the development team to build, test, and simulate flying the spacecraft long before the final assembly was completed.

Reeves's boss, Robert Anderson, broke with tradition by replacing the mainframe-based spacecraft test equipment with many interconnected single-board computers. In retrospect, the advent of microprocessors made this a logical choice. Microprocessors simplified the architecture: many independent boards reduced the likelihood of a full system failure and eliminated the single mainframe as a potential bottleneck. Reeves considers Anderson to be his most influential mentor: "He taught me that sometimes you have to step back and start with a blank sheet of paper. There's a point where capitalizing on your legacy and heritage is no longer the right path forward anymore."

After the *Magellan* project, Reeves did a brief stint outside JPL. As in many organizations, leaving and then returning is one way to increase pay and responsibility. In 1991, he came back as the leader of the team developing the software for the test equipment for the *Cassini* spacecraft. *Cassini*'s mission was to explore Saturn's rings and moons. Because *Cassini* had many more instruments than *Magellan*, Reeves extended Anderson's distributed architecture to a greater number of interfaces and simulation requirements. Each device on the spacecraft had a corresponding dedicated computer, software, and specialized hardware interface. About 20 computers were coordinated and time-synchronized. They logged data, automated testing for consistency, and provided connections to human ground operators.

Cassini was successfully launched in October 1997. In the

course of its mission, *Cassini* entered Saturn's orbit on June 30, 2004. In January 2005 a major component, the *Huygens* probe, built by the European Space Agency, dove toward one of Saturn's moons, Titan, to report on the composition of the atmosphere before it landed on the surface.

In the 1990s, NASA also initiated a series of robotic missions to Mars. It may come as a surprise to us wingless bipeds that land travel is far more difficult than flying. The challenge begins with the mechanical puzzle of landing without crushing the equipment. Once on the surface, the hardware must cope with dust, extreme temperature variations, and the difficulty of phoning home. These robotic missions clearly required a new view of design. Because of his success with prior projects, Reeves was selected to lead the team working on spacecraft software development ("flight software") for the *Mars Pathfinder* mission.

The new challenges must have seemed daunting. The spacecraft had to perform three independent missions: launch and travel to Mars, land successfully, and then act as a weather station and communication station for the *Sojourner* rover. The spacecraft had to accomplish all of this after surviving an almost crash landing on Mars and coping with limited power while being bombarded with powerful radiation.

After launch, no human can touch a spacecraft's equipment or probe it physically. Even electronic communication is fragile. "If the antenna pointed toward Earth is off a little bit, you won't hear the spacecraft, because it's transmitting with a power of what amounts to a 100-watt bulb. That's compounded by how slow it is—how many minutes it takes for a round-trip," explains Reeves. With all of these potential problems, things tend to go wrong. Fully 50% of the Mars missions have suffered failures,

and most of those failures have resulted in the catastrophic loss of the spacecraft. Many spacecraft are built with redundant hardware, and careful thought is given to the problems and failures that might develop. But not everything can be anticipated. Not one of the Mars missions has been error-free.

There are two philosophies about what to do when problems arise: (1) abandon the mission or (2) try a workaround. Reeves, the former auto mechanic, believes in anticipating problems as much as possible while building in tools to work around problems he doesn't anticipate—the "unknown unknowns," as he calls them. This philosophy comes in handy when you have to salvage a multi-million-dollar spacecraft. The fix-it philosophy also ensures that lessons learned from the problems become part of the engineering knowledge base. "If you give up, then you'll never learn anything," says Reeves. "You won't know what to fix in the future because you won't know what went wrong."

As most engineers know, there are good failures and bad ones. If there is a single root cause and it's something you could fix in the future, that's a good failure. If you can't find the root cause, then fantasies take over and you don't know what to do next. After all, the spectrum of possible causes is enormous—workmanship problems on one-of-a-kind devices, the harsh environment, dust, radiation, and aliens, to name just a few. "We haven't flown enough spacecraft to really understand the space environment. We have some knowledge, but we're still somewhat guessing," admits Reeves.

So far, the failures have been good ones; indeed, all of the failures to date could have happened on Earth. To understand both the errors and their solutions, we need to understand a bit about the debugging setup. For each spacecraft it sends into space, JPL

builds what amounts to a duplicate spacecraft on Earth. That way, activities can be practiced and commands can be verified. If the behavior of the spacecraft is abnormal, the Earth-bound mechanics have a platform for diagnosis and resolution.

Spacecraft are designed to send data and status information from time to time. If, for example, a basic "system OK" status message arrives but scientific data doesn't, there might be a problem with a scientific instrument but not with the whole device. Ground control can take an active debugging role by issuing commands that change the behavior of the spacecraft, notably by uploading software corrections or by restarting various components. Just as we often "fix" a problem on a personal computer by restarting the machine, we can do the same with spacecraft computers. As on Earth, however, restarting is sometimes not enough.

The *Mars Pathfinder* consisted of two vehicles: the *Sojourner* rover (a mobile robot designed to do experiments on the planet's surface) and the lander that provided the communication station, the imaging, and the weather station capabilities. Reeves worked on the lander. His job was to develop the software that took the spacecraft from Earth to the Martian orbit, performed the landing, and then executed activities on the surface. The rover operated independently; communication would come through the lander and be relayed to Earth.

To world acclaim on July 4, 1997, the *Pathfinder* landed on the surface of Mars, bouncing happily on its airbags. It started to collect scientific data and perform its mission. But a few days later, the lander's computer began repeatedly restarting itself, stopping all useful work. Reeves was asked to find out why. "If a computer is sitting in front of you, you hit Ctrl-Alt-Delete and you reboot.

The basic theory is the same in space," says Reeves. "The hard part is we need to know what the vehicle was doing just prior to the reset."

Fortunately, Reeves's group had designed the ability to trace what the computer was doing while it executed. They had a record of all messages sent, which significant events had occurred in the software, and, above all, which tasks were being executed and when. The debugging team* determined that a low-priority task was blocking a high-priority task. It was as if a stream of taxis had occupied an intersection and blocked an ambulance that was rushing toward a hospital.

The lander's operating system, developed by a company called Wind River, allowed higher-priority tasks to stop lower-priority tasks from running, except when the lower-priority task held a "lock" on a particular resource that the higher-priority task needed. In this case, the resource was a queue used to pass messages containing scientific data to one of the science-processing tasks. "Our first question was, Why should the low-priority task hold the lock for so long? This is where having a good test bed on Earth came in handy," notes Reeves. The team could use exactly the same software on the test bed as on the flight vehicle. Over the course of several days, they were able to cause the same problem to occur. The problem then became obvious—it was a priority inversion situation (see the box "Priority Inversion").

* The debugging team consisted of Rick Achatz, Dave Cummings, Kim Gostelow, Don Meyer, Karl Schneider, Dave Smyth, Steve Stolper, Greg Welz, and Pam Yoshioka at JPL, along with Mike Deliman, Brian Lazara, and Lisa Stanley at Wind River, the vendor of the VxWorks operating system that was used on the *Mars Pathfinder.*

licate, then shutting them down if they go rogue is not as simple as unplugging the power. They may go out of control because of numbers and replication." He maintains, however, that the positives still outweigh the negatives.

In formulating advanced concepts for space, Stoica has played with the idea of *terraforming* on other planets—creating human-friendly habitats on Mars, for example. "It's a mixture of the living and the artificial with a functional purpose." Maybe when humans arrive at a terraformed Mars, robots will form a welcoming committee. They might offer champagne. Cheers?