REPORTS & COMMENT

WASHINGTON

THE "STAR WARS" DEFENSE WON'T COMPUTE

The software problems posed by missile defense are too great for the existing computer capability

THE PENTAGON'S plans for the 1990s evoke a science-fiction film festival: robot tanks prowl battlefields, silicon co-pilots ride shotgun in dogfights, and computers scan the skies, poised to fire anti-missile missiles without human guidance or decision. The Pentagon wants "brilliant" (that is, smarter than smart) computer programs for battlefield and strategic weapons. Defense planners hope to open a new round in the arms race, one the United States can't lose. Some visionaries believe that computer science now stands at a threshold very like the one crossed by nuclear physics around 1940. They expect that in future battles the computer itself will become the most important weapon in the arsenal, while the platforms that carry it (tanks, ships, and drone aircraft) and the munitions that it discharges (conventional or nuclear) will be relegated to the status of mere accessories.

The hardware projects are ambitious, but far more so are the plans to create software—the programs, or lists of instructions, that tell the computers what to do. The Pentagon is planning the largest programs ever conceived, containing millions of instructions. Not only will these programs be larger than any now in existence but they are supposed to transcend the rigid routines of today's programs and attain a flexibility approaching human thought. Computers with these characteristics represent the yet-to-appear "fifth generation" of computer technology—they exhibit "artificial intelligence."

The technologies involved in these computers do not yet exist, and whether or not they ever will is controversial. Yet they are vital to the success of several costly weapons programs, most notably the $26 billion Strategic Defense Initiative (SDI), better known as the Star Wars missile defense. Robert Cooper, the head of the Defense Advanced Research Projects Agency (DARPA), the Pentagon's main agency for basic research, promises that the new computers will be "an enabling technology for a defense as complex as may be necessary for ballistic missile defense." Many computer scientists are skeptical, however. They think that we are betting the security of the country on hypothetical discoveries that may never occur. The Pentagon alludes to these as basic research projects, not applied research to develop weapons for the field. But we are already concentrating our research resources on them and are therefore abandoning alternatives. In the future we may find that we have become committed by default and that we have to use the results of these projects, no matter how far short of today's promises they may fall.

The Department of Defense (DoD) is now subordinating computer science to military needs as completely as nuclear physics, aeronautics, and rocketry were subordinated in the 1940s. An unprecedented flow of DoD dollars is intended, in the Pentagon's words, to "push" and "pull" the nation's computer scientists into working on "carefully selected military applications." At a time when Japan is funding an effort of similar scope to dominate the commercial computer market (about which more later), the wisdom of militarizing computer science is doubtful. The Pentagon admits that "the magnitude of this national effort could represent a very large per-

ILLUSTRATIONS BY JOSÉ CRUZ

turbation to the university community"; nonetheless, there has been little opposition from computer scientists. A few hope for a Pax Americana guaranteed by uncontested technological prowess rather than by the traditional sorts of military power. Others are skeptical, but most are indifferent to the political implications of the bottomless well of defense-contract money for their research projects and businesses.

So the work begins. An aura of naive overconfidence hangs over much of the effort. The proposals and progress reports suggest attitudes shaped in seminars on the campus of a suburban defense think tank, not on the battlefield or training ground. They reveal a fascination with abstruse theory and a delight in virtuoso puzzle-solving coupled with a nonchalance toward practical problems and a flip disregard for safety and reliability. While DoD is attempting to persuade the public that infallible robot warriors will remove the risk and uncertainty from combat and permit Americans to wage wars without casualties, the race to close the incipient robot gap is already on. At a Moscow trade fair last fall the Moscow Academy of Sciences announced a five-year, $100 million program by Soviet and Warsaw Pact scientists to develop fifth-generation computer technologies for the Eastern bloc.

FIFTH-GENERATION computing technology is slated for application in many DoD programs, but the most extreme examples occur in something called the Strategic Computing Program (SCP). This five-year, $600 million project was announced in October of 1983 by DARPA, and it got under way last year, with a first-year budget of $50 million.

The central goal of the project is to apply a family of programming techniques called "artificial intelligence" to a new generation of brilliant weapons that would be able to perform complex missions without human guidance. It is described in the DARPA report *Strategic Computing, New-Generation Computing Technology: A Strategic Plan for Its Development and Application to Critical Problems in Defense*. If the goal is even partially achieved, future historians might rank this remarkable document with Albert Einstein's 1939 letter to President Roosevelt recommending development of the atomic bomb. Its peculiar tone, at once extravagant and vague, is difficult to convey in paraphrase. It begins:

> Instead of fielding simple guided missiles or remotely piloted vehicles, we might launch completely autonomous land, sea, and air vehicles capable of complex, far-ranging reconnaissance and attack missions. . . . In contrast with previous computers, the new generation will exhibit human-like, "intelligent" capabilities for planning and reasoning. . . . Using this new technology, machines will perform complex tasks with little human intervention, or even with complete autonomy. . . . Our leaders will employ intelligent computers as active assistants in the management of complex enterprises.

The report concludes that "the possibilities are quite startling, and could fundamentally change the nature of future conflicts."

The SCP pilot projects already under way are various. For the Army there is an "autonomous land vehicle," a sort of unmanned robot tank, which would be capable of roving independently and would be entrusted with such missions as "deep-penetration reconnaissance . . . and weapons delivery." For the Air Force there is the "pilot's associate," a silicon co-pilot to be installed in a fighter aircraft to help operate the electronic-warfare equipment and to impart "instruction on advanced tactics from more experienced pilots to aid the less experienced pilot on his first day of combat." A third project is to be a computer system aboard the aircraft carrier USS *Carl Vinson* devoted to intelligence analysis, to help the Navy fight battles at sea. DARPA expects that the lessons learned in these projects will be applied in a far more ambitious undertaking, an automated ballistic-missile defense system—Star Wars.

Most criticism of Star Wars has concentrated on the physical difficulties of destroying large numbers of missiles in flight, but there are unsolved problems of similar magnitude in controlling such a system, even if one could be developed. In a September, 1983, interview in *Omni* magazine, Defense Secretary Caspar Weinberger described the problem and explained his confidence in computer technology as the answer.

> WEINBERGER: The goal would be to try it against thousands of missiles, including missiles that carry ten independent warheads, and missiles whose warheads can change direction. It is, I am told, essentially a problem in very, very large and extraordinarily rapid computer capability. We must develop that to the point where we can reliably identify, track, and destroy several thousand targets in a very, very short space of time.
>
> OMNI: You are talking about a total battle time of as little as possibly one hundred twenty or two hundred seconds?
>
> WEINBERGER: It is very short. It is a very big task—a task about which a lot of people say, "Well, we can't do it." But then, a lot of people said that we couldn't fly.

The missile-defense proponents appeal to computers for the solutions to all sorts of unresolved difficulties. In response to the comment of one critic, Richard Garwin, that an adversary could cheaply foil such a defense by filling the sky with decoys, the President's science adviser, George Keyworth, told *Omni*, "The thing to focus on is data processing. If you can handle enough data fast enough, you can do a pretty darn good job of discrimination."

The most startling aspect of the missile-defense proposals is that the decision to fire must be made automatically. Analysts agree that given the speed of incoming missiles there is no question of human observers even participating, much less notifying their commanders and waiting for orders. DoD's Fletcher panel, which analyzed the proposed Star Wars system, claimed, "It seems clear . . . that some degree of automation in the decision to commit weapons is inevitable if a ballistic missile defense system is to be at all credible." The computers would make the judgment that an attack was in fact under way, based on information (itself processed by computer) from observation satellites and perhaps from ground-based radars. This expectation is explicitly spelled out:

> Commanders remain particularly concerned about the role autonomous systems would play during the transition from peace to hostilities when rules of engagement may be altered quickly. An extremely stressing example of such a case is the projected defense against strategic nuclear missiles, where systems must react so rapidly that it is likely that almost complete reliance will have to be placed on automated systems.

DARPA's intent to delegate to computers the authority to fire missile interceptors was reiterated by Keyworth and by Robert Cooper, before the Senate Foreign Relations Committee in April of 1984. As reported by the *Seattle Times*, they testified that the proposed systems

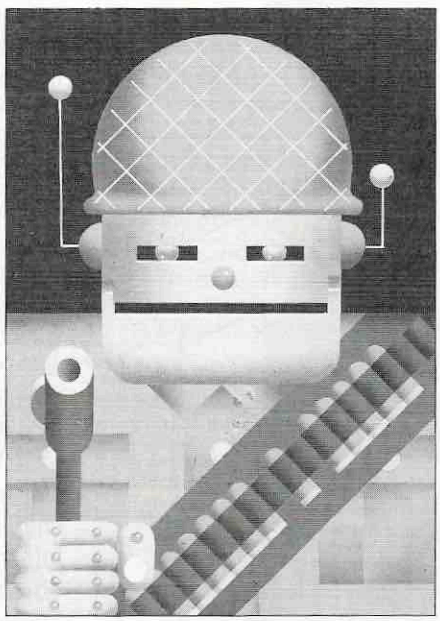
would need to be triggered on extraordinarily short notice.

> "Who's going to make that decision?" said [Senator Paul] Tsongas.
> "We don't know," said Keyworth. "By the year 1990, it may be done automatically."

Later, according to an Associated Press account, "Sen. Joseph Biden, D-Del., pressed the issue over whether an error might provoke the Soviets to launch a real attack. 'Let's assume the president himself were to make a mistake . . . ,' he said. 'Why?' interrupted Cooper. 'We might have the technology so he couldn't make a mistake.' " The underlying assumption here seems to be that with refinements and elaborations the computers within warning-and-launch systems could replace human observers and decision-makers—whose judgments do not now depend completely on the correctness and reliability of those computers. Experience with existing military computer systems suggests that this is a fundamental misconception, potentially a mortally dangerous one.

Administration weapons spokesmen have considered the possibility that the missile-defense computer might fail. Keyworth and Lieutenant General James Abrahamson, the director of the Strategic Defense Initiative, have indicated that the computers might be turned on only during crises. Abrahamson even speculated that we might warn the Russians when this occurs, so that they could be sure to be extra careful. Last year he told the House Appropriations Committee, "Under a crisis decision you would probably escalate this in the way that says if there is no crisis many of the automatic features would be shut down and turned off. But were a crisis to build, the President would make those decisions ahead of time, and he would probably tell the Russians, 'Okay, now I am activating an important part of our system,' which means that it is automatic. Hopefully that would begin to reduce the crisis."

These comments reveal how untenable the proposals really are. Keyworth's and Abrahamson's suggestions that the system would be activated only during crises strongly imply that even they do not believe that the system could be made very reliable: if we leave it turned off most of the time, we might reduce the chance of an accident. But many analysts have observed that a crisis is exactly the worst time to delegate control to an unstable system. The chances that there will be some incident that triggers a false alert are probably greater then, and the consequences of responding to a false alert could be much worse. Alan Borning, a researcher in artificial intelligence at the University of Washington, has written about our present-day systems, "During times of relative international calm, the combined U.S.–Soviet system probably has enough human checks . . . to cope with a single mechanical or operator error. . . . The situation would be different during a time of great tension or conventional war. Under such circumstances, the officers monitoring the systems would be less ready to dismiss a warning as being the result of a



computer error, and the danger of escalating alerts on each side would be much greater."

EXPERIENCE TEACHES that we cannot rely on computers in situations of great uncertainty. Yet these are the situations in which DARPA hopes to rely on autonomous machines. Rather than commanders "forced to rely solely on their people to respond in unpredictable situations," DARPA hopes to have supercomputers that will "provide more capable machine assistance in unanticipated combat situations." The error here is a fundamental one, but DARPA proposes to patch it over by simply adopting some new techniques, most of which are borrowed from artificial intelligence (AI).

As the name implies, this is a field with almost mythic ambitions: the creation of computer programs that imitate the human mind. If such protean programs really existed, one might argue that they should be exempt from the human supervision required for lesser machines. DARPA presumes that they will soon exist. In fact AI is an immature field, still far short of its ambitious goals, although, as the computer scientist Severo Ornstein and Brian Smith and Lucy Suchman, of the Xerox Palo Alto Research Center, have written, "Shortcomings are often masked by subtle semantic shifts. When we fail to instill 'reasoning' or 'understanding' in our machines, we tend to adjust the meaning of these terms to describe what we have in fact accomplished." Ornstein says, "As a concept, AI is like jelly—when you push on it in one place, it just goes someplace else. It is really just a term we apply to problems that seem intractable. Once they are thoroughly solved, they are not AI anymore but just another computer program."

Artificial-intelligence programming has been done mostly in university laboratories, not at defense plants. Because of the different needs and goals in these two environments, two quite different programming cultures have grown up. On the one hand, most of DoD's programmers work with "embedded" computers (computers included in larger systems, such as weapons or vehicles) destined for the field, are strongly oriented toward issues of reliability and maintenance, and try to build systems whose behavior is tightly specified in advance. These values prevail in other arenas, such as medicine and industrial applications other than weapons, where safety is an important consideration. Programmers who hold these values often refer to themselves as "software engineers."

The university programmers oriented toward artificial intelligence, on the other hand, favor open-ended research. They have no need for highly reliable systems, so dependability has never been a priority for them, and a decidedly casual attitude toward errors prevails. This group is sometimes described as "hackers." (The term is much older than its recent connotations of juvenile computer crime, and derives from the usual meaning of *hack*, "to cut irregularly, without skill or definite purpose.") It is this group that DARPA plans to enlist in the Strategic Computing Program.

THE GLAMOUR FIELD of the eighties in computer science is definitely AI. Some researchers believe that discoveries imminent in AI will permit computers to recognize objects in video images and converse with humans in natural language, making it possible to build robots much like those of the popular imagination. Others are more conservative but expect the research to yield results of great practical and commercial value. AI's vast potential has been vigorously promoted to the public and to the business community, through popular magazine articles and books, including Edward Feigenbaum and Pamela McCorduck's *The Fifth Generation*. As a result there has been a great deal of investment, or at least talk of investment, in the field. Some economists and business leaders agree with Feigenbaum and McCorduck that in the near future prowess in AI will replace pretty much everything else as the vital ingredient in a nation's economic health.

Supporting AI, therefore, has become practically a patriotic duty. DARPA played heavily on this sentiment to launch its Strategic Computing Program. But the real boost for AI in this country came with the news that the Japanese had launched their major national effort to exploit AI in the commercial marketplace. In October of 1981 the Japanese Ministry of Trade and Industry announced a ten-year, $850 million Fifth Generation Computer Systems Project, to be carried out at its Institute for New Generation Computer Technology, with the assistance of industrial giants like Hitachi and Mitsubishi. The idea is to apply the fruits of research accomplished at the institute throughout Japanese industry. The announcement frightened U.S. computer firms. Already dominant in consumer electronics, Japan was also competitive in the integrated-circuit components used in all computers. Now it was preparing to take over computer design, software, services—everything.

At about the same time, DARPA was seeking support in Congress for what was to become the Strategic Computing Program (then called the supercomputer project). DARPA staffers began putting out the word on Capitol Hill that this would be the U.S. response to the Japanese Fifth Generation. "I don't think $50 million is enough," said Anthony Battista, a House Armed Services Committee staff member, about DARPA's first-year funding request. He added, "[The supercomputer is] a problem that goes far beyond the Defense Department; it trends directly into our whole economic base." A few days later a favorable editorial appeared in *The Washington Post*. The SCP was funded.

But whereas the Japanese project is intended to enhance business and consumer productivity and to improve social services, Mark Stefik, a scientist at Xerox, notes, "the American plan is rationalized by military needs, and secondarily commercial values."

In fact, by concentrating on military applications the United States may be handicapping itself in the commercial race. For example, some SCP discoveries may be unavailable for commercial development. Ed Zschau, a Republican congressman representing a Silicon Valley district, has said, "We may find that the technology gets bottled up inside the military establishment. My concern is that specific technological breakthroughs . . . may be classified . . . and therefore may be difficult to get out into the private sector."

Even worse, the SCP will drain talent away from commercial development projects. DARPA denies that the problem exists, making the peculiar argument that business is not interested in developing advanced computer applications anyway. In February of 1984 Robert Cooper told a meeting of electronics engineers that backing the SCP is "buying an insurance policy against IBM management's decision not to pursue machine intelligence through the 1990s. They have abandoned supercomputers, believing that such investments don't yield a good payoff." Richard DeLauer, a former undersecretary of defense for research and engineering, has said, "The Defense Department should press this technology, because no one else is pursuing it, and the Japanese have strong programs." This is simply not true. The real U.S. response to the Japanese effort lies in commercial consortia like the new Microelectronics and Computer Technology Corporation (MCC). Launched in January, 1983, with a $75 million yearly budget, by industry giants like Control Data, Digital Equipment, Motorola, and others (not including IBM), MCC is headed by retired Admiral Bobby Ray Inman, a former director of the National Security Agency and a former deputy director of the CIA. Of the SCP he says, "The real problem is that ultimately we'll be competing for the same talent."

WILL WE REALLY be able to place "complete reliance" on autonomous computers to perform critical functions when the stakes are high? Probably not. Almost every computer scientist I have spoken with says that DARPA grossly understates the difficulty of the project. Its proposal assumes that the goal can be achieved simply by scaling up present computers with smaller transistors, faster processors, more rules. But the scientists know that fundamental theoretical breakthroughs will also be required. Since these have proved elusive for decades, they are unlikely to appear on schedule.

There are other obstacles as well. Every one of today's large computer systems contained undiscovered flaws that revealed themselves only after the systems were placed in use. DoD has been grappling with this problem for years and has achieved some results, but a complete solution is nowhere in sight. Star Wars and the SCP will require major advances in the practice of building error-free hardware and software. The projects presume that the problem will be solved in the next few years. For DARPA, this literally goes without saying—the 90-page SCP proposal says nothing about testing, reliability, or quality assurance. Furthermore, the solution to this problem has nothing to do with such technical feats as packing more transistors on a microchip. Computer errors are as much an administrative and behavioral problem as a technical one.

There are many kinds of computer errors. Among the easiest to understand are hardware failures—the computer simply breaks. The false alerts about Soviet missile attacks in June of 1980 were traced to the failure of a silicon chip in a communications computer at the North American Aerospace Defense Command (NORAD). Also, computer operators can make mistakes. A similar alert happened in November of 1979, when a training tape containing a simulation of a Soviet attack was somehow played through NORAD's "live" warning system. Or programs may be provided with incorrect data. In a 1979 airliner tragedy incorrect routing data were entered into an Air New Zealand navigation computer. On the way to Antarctica in bad weather, the plane was twenty-seven miles off course. It crashed into Mount Erebus, and more than 200 people were killed.

But many computer errors are not so easy to understand. When the military

began installing computers in weapons, this introduced a fundamentally new kind of problem, never encountered in simpler weapons: programming errors, or, as programmers say, "bugs."

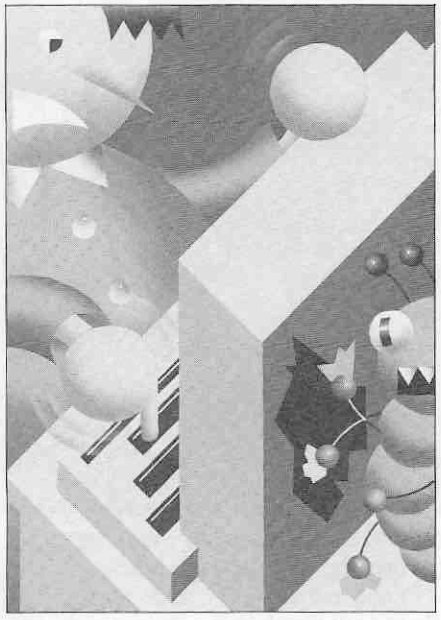
A program's size is measured by counting the number of programming-language statements, or "lines of code," it contains. For relatively simple devices, like microcomputer-controlled video games and home appliances, the programs may be a few hundred or at most several thousand lines long. For a complex weapons system like a guided missile, the programs typically run to tens or hundreds of thousands of lines, distributed among several dissimilar devices: the launcher, the tracking radars, and the missile itself. And if the programs contain mistakes, the missile will not work.

On projects of this kind, writing the software is often the most expensive portion of the whole development effort. Almost every weapons system fielded today contains one or more computers, from microcomputers built into projectiles to large mainframes behind the lines. In 1984 DoD spent $11 billion on embedded-computer systems. Of this, $8.5 billion—roughly 80 percent—went for software. The effort that went into developing and testing the software for the F-18 fighter was so extensive that when changes are required, the plane is modified to fit the existing software, rather than the software to fit the plane. DoD expects that by 1990 it will spend more than $30 billion a year, or about ten percent of the entire defense budget, for software.

DoD is proposing the most difficult programming efforts yet attempted. According to the department's own estimates, the Star Wars programs alone will comprise about ten million lines of code. DoD admits that "developing hardware will not be as difficult as developing appropriate software" but promises that the programs will "operate reliably, safely, and predictably."

In fact nobody knows how to develop such programs. As programs grow, it becomes disproportionately more difficult to ensure that they are correct. A ten-thousand-line program is much more than ten times as difficult to debug as a one-thousand-line program, because the pieces may interact in subtle ways. Further, when more programmers must be put to work for a longer time, there are additional opportunities for misunderstandings. As Alan Borning explains about today's large programs, "Usually there is *nobody* who understands the entire system completely." In a standard programming textbook Edward Yourdon, a software-reliability expert, characterizes proposals to build programs more than a million lines long as "utterly absurd," and adds,

> It is the author's opinion that our current knowledge of program design and testing concepts is not sufficient to successfully produce a system of such size. . . . In some cases, the success of the system is perhaps a moot point, e.g., an air defense system that will hopefully never be used; on the other hand . . . the consequences of a poorly tested system are rather staggering.



Bugs take many forms. Each stage in the program-development process introduces a corresponding species of error. The simplest, such as typographical slips, can have large consequences. An often repeated bit of programmers' folklore has it that the failure of the 1962 *Mariner I* probe to Venus, which had to be blown up shortly after launch, was caused by the substitution of a period for a comma in a statement in the infamous FORTRAN language. Most modern programming languages automatically check for simple mistakes in notation but can do little to prevent errors in logic that may be introduced during the design stage.

The subtlest errors occur when the programmer forgets to consider, or is totally ignorant of, certain contingencies that the program might encounter. These errors are the toughest to root out, because there is no mistake to be found: the error lies in what is *not* present. They are also inevitable; programmers are no better than anyone else at foreseeing the future. One spectacular example was revealed after the sinking of the British destroyer *Sheffield* in the Falklands war. The *Sheffield*'s detection equipment picked up the incoming Argentine missile and identified it as an Exocet. But the Royal Navy, which was geared to fighting the Soviet Union rather than a country operating Western weapons, had not programmed the computer to recognize the French-built Exocet as hostile.

Programming errors are uniquely insidious, because they give no overt evidence of their existence—there is nothing "broken" to discover in any sort of inspection, and the program text itself is frequently unavailable, or incomprehensible to anyone but its authors. A bug reveals itself only when some event activates the portion of the program where it lurks. Programmers try to enter code without bugs, and for some kinds of problems it is even possible to prove that a program is correct, in much the same way that theorems in geometry are proven. But in most practical situations errors can be uncovered only by testing. With very complex programs, the number of contingencies is so large that not every possibility can be tested. And however conscientious and thorough the testers may be, they cannot test a program's performance in contingencies that they cannot foresee.

As a result, *all large programs contain undiscovered errors and omissions that come to light only after prolonged experience in actual use.* This is the single most important fact for users of complex computer systems to understand. To use a computer responsibly, one must not trust it. One must plan in advance how to detect the inevitable errors when they come up, and how to recover from them. I write computer programs that help plan radiation-therapy treatments for cancer. The programs calculate the radiation dose that a proposed treatment would deliver to each of several hundred locations in the body. For each treatment my co-workers and I also estimate the doses to a few locations by another method, using tables and a hand calculator. Discrepancies occasionally appear. Once, a team at another hospital discovered an error in a program we had been using for more than two years; the particular combination of circumstances that revealed

it had never occurred in our department.

In the programming business the critical issue is "support." A software house provides support if it collects bug reports from its customers, fixes the problems, and distributes revised programs. Some shops circulate periodic "system dispatches" to customers, describing recently discovered bugs and recommending work-arounds and recovery procedures pending distribution of the repaired programs. This kind of work is called software maintenance. It is never-ending, because customers demand improvements to handle situations and new uses that the authors did not originally consider, and these modifications (as well as the changes that were supposed to fix earlier errors) also contain errors, and so on.

The only evidence most people have of all this activity is the inevitable billing mistakes, lost reservations, and other problems that occur when any large organization computerizes its operations. Many people assume, though, that for critical military applications programmers use an especially rigorous testing regimen that enforces a higher standard than is normally encountered in civilian life. They don't.

This disturbing truth is revealed by the occasional reports of military computing accidents that reach the press, such as the false NORAD alerts and a 1983 naval exercise in which a computer-controlled gun turret fired in the opposite direction from its intended target, toward shipping lanes. Some of the problems have consequences more serious than single incidents; they pervade an entire system and require expensive redesign. According to a story I heard among programmers in the aerospace industry in the early 1970s, a serious flaw in the Minuteman missile-guidance programs, which had somehow been discovered, required the reprogramming of missiles already in their silos. In a 1977 test of the Worldwide Military Command and Control System, attempts to send messages failed 62 percent of the time. One part of the network, the Readiness Command, broke down 85 percent of the time. Such communications breakdowns have been implicated in the capture of the USS *Pueblo* by North Korea in 1968 and in the Israeli bombing of the U.S. intelligence ship *Liberty* during the Six-Day War.

Proponents of the most farfetched weapons schemes always answer skeptics with "Well, we got to the moon, and

people said we couldn't do that either." But each space mission is accomplished with a concentration of talent, an absence of time pressure, an abundance of resources, and a secure environment—a combination that could never be replicated in combat. And even under optimal circumstances unforeseen problems appear. The first shuttle flight, in 1981, was delayed when a bug appeared only twenty minutes before the launch was to take place. It was eventually traced to a programmers' failure to understand the implications of a program fix that had been made a year earlier to correct the effects of another modification made a year before *that*. Another problem appeared during rehearsal for the second shuttle mission. The crew aborted the simulated flight; then they decided to "return to Earth" even earlier and tried to abort the abort. They couldn't. Evidently the programmers had not foreseen that anyone might want to abort the same mission twice. Even in the optimal environment of space research, some bugs are discovered only after the programs are put to use.

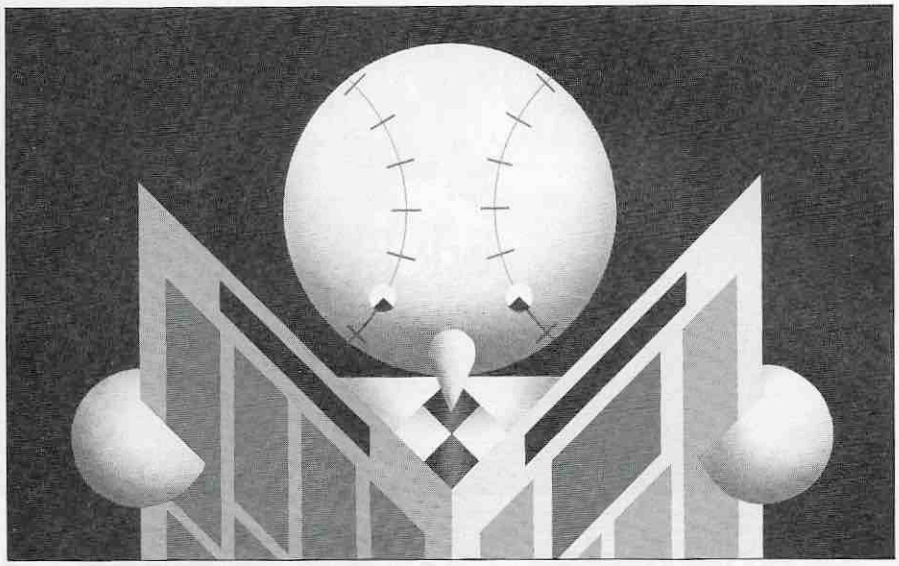
And what of the most critical application of all? Many people believe that already NORAD and the Strategic Air Command would make the decision to launch nuclear missiles solely on the basis of computer-generated data. They wouldn't. Or at least they didn't in 1979 and 1980, because the false alerts that occurred then were discovered and negated before any harm was done. In the words of the Senate report on the incidents, "Even though the mechanical electronic part produced erroneous information, the human part correctly evaluated it and prevented any irrevocable reaction." The skill of NORAD's human part is so highly evolved because NORAD has been working with complex computers longer than almost any other organization—since the installation of the SAGE air defense system, in the late 1950s. The staff has had to adjust and improvise to keep the system running. When DARPA proposes autonomous "battle-management systems," it is announcing its belief that we will no longer require the sort of improvisation that enabled NORAD to cancel the false alerts.

The people who design and program these systems are neither stupid nor negligent. The problems arise because the task is intrinsically difficult and the expectations for the technology are unreasonably high, given the realities of combat.

No matter how many megabytes of memory or millions of lines of code DARPA packs into its supercomputers, the fundamental fact about computer programs will not change: it is impossible to find all the bugs by analysis and testing alone—the program must be used under actual conditions. We cannot know how well the robots work until we send them to war. No doubt there will be opportunities to try out robot tanks and other tactical weapons in some unfortunate Third World or Middle Eastern country. As for missile defense, Ira Kalet, a University of Washington researcher in medical applications for AI, asks, "Do we understand and have experience with nuclear warfare? Do we want to gain this experience?"

—*Jonathan Jacky*

Jonathan Jacky is a research assistant professor in the Department of Radiation Oncology at the University of Washington School of Medicine. He is the author of numerous scholarly articles on computer science.



NOTES

A WHOLE DIFFERENT BALL GAME

"There are three things the average man thinks he can do better than anybody else: build a fire, run a hotel and manage a baseball team."

—Rocky Bridges,
manager, the San Jose Bees

NOT LONG AGO, as spring training was about to commence, I had breakfast at a fashionable Washington hotel with the energetic young owners of a local baseball franchise. They talked with refreshing candor about the season just past and the one soon to start.

"Compared with everybody else in the league, I don't think that either Brian or I was too knowledgeable about baseball," Mark London, a Washington lawyer, admitted. "I mean, we knew how to puff cigars and hang tough and negotiate, and that's how we got Rickey Henderson. But we weren't really knowledgeable about baseball. At least not this kind."

"I think we'll smoke Churchills at the next draft," added his co-owner, Brian Kelly, a journalist who lives in Chicago. "We're just going to sit there and blow smoke, and it will drive the others crazy."

"We did come pretty close to the money, though," London continued. "The entire year we hovered around the money, and in the end we were only two RBIs away. *Two RBIs.* You know what two RBIs is? Two RBIs . . ."

"Two RBIs," Kelly interrupted, "is a single with the bases loaded in a game that gets rained out and doesn't count."

London and Kelly own the Washington, D.C., Amazons, which they acquired in 1984. Last year the Amazons finished behind five teams but ahead of six others in the Washington Ghost League. That is a respectable effort for a young ball club, but Kelly and London believe that the Amazons came closer to the pennant than the final standings suggest. They concede that their own inexperience proved costly. Key trades fell through or were somehow sabotaged by the competition. The Amazons were plagued as well by untimely injuries. Prolonged slumps beset several ballplayers. But just as troublesome was the inherent quirkiness of the game—Rotisserie League baseball—that the twelve clubs in the Washington Ghost League play.

"We had, on paper, a team with a tremendous amount of talent," London explained. "We had Rickey Henderson, who is one of the greatest players in baseball. For seventy cents we got Andre Thornton, who had thirty-three homers and ninety-nine RBIs last year. We had Mike Easler, the Hit Man, who batted .313 and knocked in ninety-one runs. We had people like Gary Pettis, who we bought for twenty cents and, as it turned out, was the third leading base stealer in our league. For a very low price we got a guy named Gary Ward—a tremendous home-run hitter for Texas who batted .600 in the second half of the season."

"We had Aging All-Star Don Baylor," Kelly said.

"Right, we even had Aging All-Star Don Baylor. But unfortunately some of the big guys didn't do what they were supposed to do. You pay them all that money and they still don't come through for you. We bet and lost on "Clams" Castino, who is a really fine second baseman for the Twins. He cost us ninety cents, and then he hurt his back. Tim Hulett is a Chicago infielder whom we needed to fill a hole, and the Sox ended up sending him to the minors after a month. For a dollar we signed Dave Stegman, of the White Sox, who had shown some promise again, and Stegman immediately turned around and went south."

"A complete nose dive," Kelly said. "I'd seen some of the stats on Stegman during spring training, and he made a couple of appearances at the beginning of the season, looking good. I talked him over with Mark a couple of times and then, when we needed to fill another hole and I wasn't around, Mark drafted the guy as a kind of present, thinking, 'Brian's in Chicago, maybe we ought to have a Chicago guy he can watch on television.'"

"Roy Smalley was another disaster," London said.

"Smalley cost us," Kelly agreed. "Smalley hurt us badly."

"And we made some mistakes, like taking Teddy Simmons as a designated

JUNE 1985

$2.00

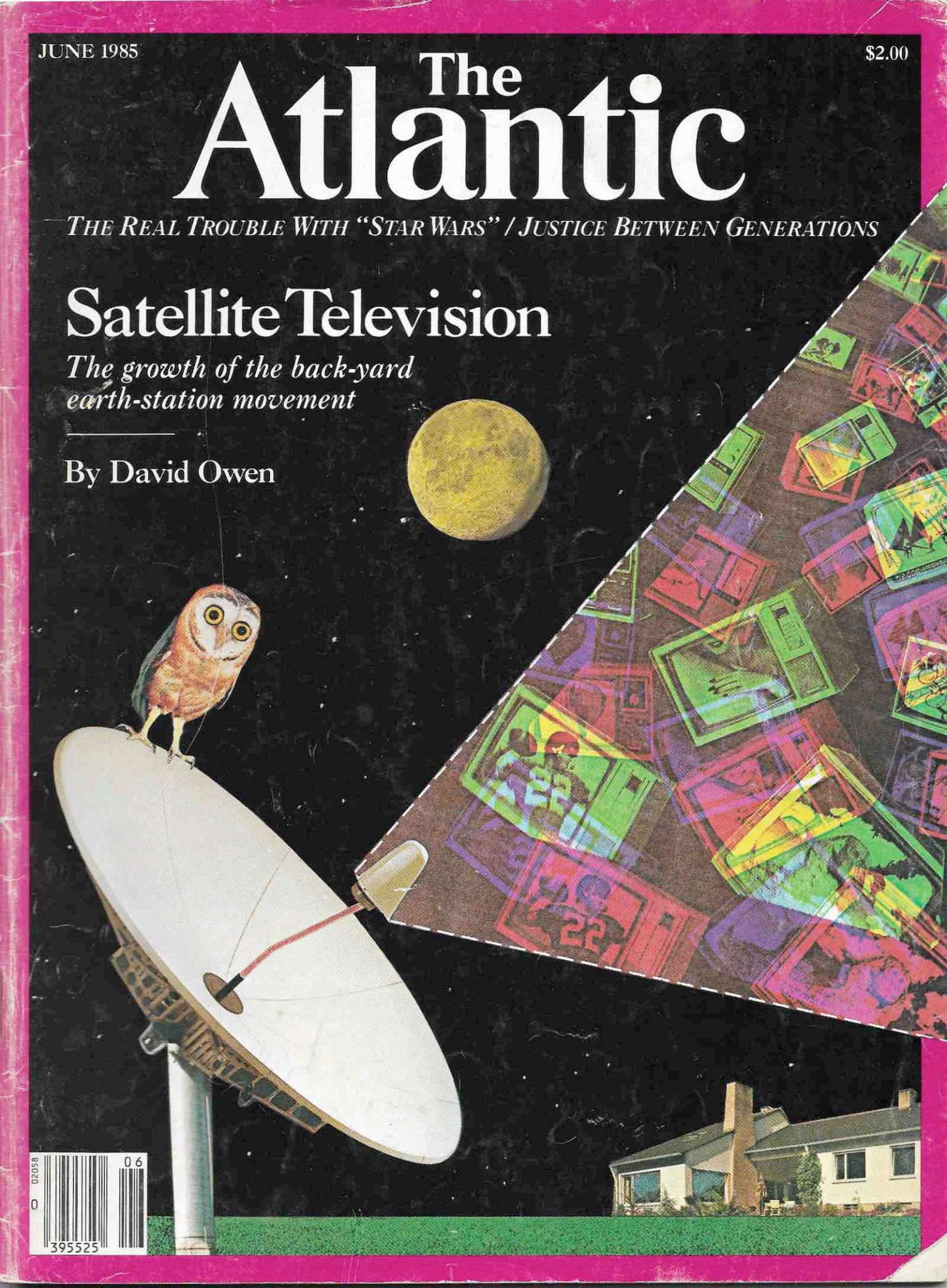
The Atlantic

THE REAL TROUBLE WITH "STAR WARS" / JUSTICE BETWEEN GENERATIONS

Satellite Television

The growth of the back-yard earth-station movement

By David Owen



0 02058 06 395525