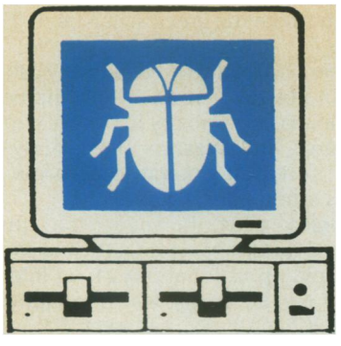


Finding Fault

The formidable task of eradicating software bugs

By IVARS PETERSON

Sitting 70 kilometers east of Toronto on the shore of Lake Ontario, the Darlington Nuclear Generating Station looks much like any other large nuclear power plant of the Canadian variety. But behind its ordinary exterior lies an unusual design feature.

Darlington is the first Canadian nuclear station to use computers to operate the two emergency shutdown systems that safeguard each of its four reactors. In both shutdown systems, a computer program replaces an array of electrically operated mechanical devices — switches and relays — designed to respond to sensors monitoring conditions critical to a reactor's safe operation, such as water levels in boilers.

When completed in 1992, Darlington's four reactors will supply enough electricity to serve a city of 2 million people. Its Toronto-based builder, Ontario Hydro, opted for sophisticated software rather than old-fashioned hardware in the belief that a computer-operated shutdown system would be more economical, flexible, reliable and safe than one under mechanical control.

But that approach carried unanticipated costs. To satisfy regulators that the shutdown software would function as advertised, Ontario Hydro engineers had to go through a frustrating but essential checking process that required nearly three years of extra effort.

"There are lots of examples where software has gone wrong with serious consequences," says engineer Glenn H. Archinoff of Ontario Hydro. "If you want a shutdown system to work when you need it, you have to have a high level of assurance."

The Darlington experience demonstrates the tremendous effort involved in establishing the correctness of even relatively short and straightforward computer programs. The 10,000 "lines" of instructions, or code, required for each shutdown system pale in comparison with the 100,000 lines that constitute a typical word-processing program or the millions of lines needed to operate a long-distance telephone network or a space shuttle.

Computer programs rank among the most complex products ever devised by humankind, says computer scientist David L. Parnas of Queen's University in Kingston, Ontario. "They are also among the least trustworthy," he contends.

"These two facts are clearly related," says Parnas. "Errors in software are not caused by a fundamental lack of knowledge on our part. In principle, we know all there is to know about the effect of each instruction that is executed. Software errors are blunders caused by our inability to fully understand the intricacies of these complex products."

Practically no one expects a computer system to work the way it should the first time out. "A new chair collapses, and we're surprised," Parnas says. In contrast, "we accept as normal that when a computer system is first installed, it will fail frequently and will only become reliable after a long sequence of revisions."

But there are many situations where that kind of performance is unacceptable. Computers that fly military or civilian aircraft, operate medical devices, manage transportation systems and perform crucial safety functions such as air-traffic control must work without fail.

As computer-controlled systems increase in complexity and become ever more deeply embedded in the fabric of society, the potential for costly failures rises. Indeed, some computer experts fear that we are courting disaster by placing too much trust in computers to handle complexities that no one fully understands.

Last November, the Association for Computing Machinery sponsored a meeting in Arlington, Va., on the issue of managing complexity — finding ways to build computer systems that are both large and trustworthy. "There are tons of issues out there," says Harold S. Stone of the IBM Thomas J. Watson Research Center in Yorktown Heights, N.Y. "This is one we can't get our finger on. We don't have the answer."

Companies throughout the computer and communications industries, including giants such as IBM and AT&T, are having great difficulties developing the next generation of computer products, Stone adds. "We need to go to the next [higher] plateau in automation, and we can barely deal with the plateau that we're on now."

Two case histories — testing the safety software for Ontario Hydro's Darlington plant and a software error that nearly crippled AT&T's long-distance network — nicely illustrate this point.

The software glitch that disrupted AT&T's long-distance telephone service for nine hours in January 1990, dramatically demonstrates what can go wrong even in the most reliable and scrupulously tested systems. Of the roughly 100 million telephone calls placed with AT&T during that period, only about half got through. The breakdown cost the company more than $60 million in lost revenues and caused considerable inconvenience and irritation for telephone-dependent customers.

The trouble began at a "switch" — one of 114 interconnected, computer-operated electronic switching systems scattered across the United States. These sophisticated systems, each a maze of electronic equipment housed in a large room, form the backbone of the AT&T long-distance telephone network.

When a local exchange delivers a telephone call to the network, it arrives at one of these switching centers, which can handle up to 700,000 calls an hour. The switch immediately springs into action. It scans a list of 14 different routes it can use to complete the call, and at the same time hands off the telephone number to a parallel, signaling network, invisible to any caller. This private data network allows computers to scout the possible routes and to determine whether the switch at the other end can deliver the call to the local company it serves.

If the answer is no, the call is stopped at the original switch to keep it from tying up a line, and the caller gets a busy signal. If the answer is yes, a signaling-network computer makes a reservation at the destination switch and orders the original switch to pass along the waiting call — after that switch makes a final check to ensure that the chosen line is functioning properly. The whole process of passing a call down the network takes 4 to 6 seconds. Because the switches must keep in constant touch with the signaling network and its computers, each switch has a computer program that handles all the necessary communications between the

switch and the signaling network.

AT&T's first indication that something might be amiss appeared on a giant video display at the company's network control center in Bedminster, N.J. At 2:25 p.m. on Monday, Jan. 15, 1990, network managers saw an alarming increase in the number of red warning signals appearing on many of the 75 video screens showing the status of various parts of AT&T's worldwide network. The warnings signaled a serious collapse in the network's ability to complete calls within the United States.

To bring the network back up to speed, AT&T engineers first tried a number of standard procedures that had worked in the past. This time, the methods failed. The engineers realized they had a problem never seen before. Nonetheless, within a few hours, they managed to stabilize the network by temporarily cutting back on the number of messages moving through the signaling network. They cleared the last defective link at 11:30 that night.

Meanwhile, a team of more than 100 telephone technicians tried frantically to track down the fault. By monitoring patterns in the constant stream of messages reaching the control center from the switches and the signaling network, they searched for clues to the cause of the network's surprising behavior. Because the problem involved the signaling network and seemed to bounce from one switch to another, they zeroed in on the software that permitted each switch to communicate with the signaling-network computers.

The day after the slowdown, AT&T personnel removed the apparently faulty software from each switch, temporarily replacing it with an earlier version of the communications program. A close examination of the flawed software turned up a single error in one line of the program. Just one month earlier, network technicians had changed the software to speed the processing of certain messages, and the change had inadvertently introduced a flaw into the system.

From that finding, AT&T could reconstruct what had happened.

The incident started, the company discovered, when a switching center in New York City, in the course of checking itself, found it was nearing its limits and needed to reset itself – a routine, maintenance operation that takes only 4 to 6 seconds. The New York switch sent a message via the signaling network, notifying the other 113 switches that it was temporarily dropping out of the telephone network and would take no more telephone calls until further notice. When it was ready again, the New York switch signaled to all the other switches that it was open for business by starting to distribute the calls that had piled up during the brief interval when it was out of service.

One switch in another part of the country received its first message that a call from New York was on its way, and started to update its information on the status of the New York switch. But in the midst of that operation, it received a second message from the New York switch, which arrived less than a hundredth of a second after the first.

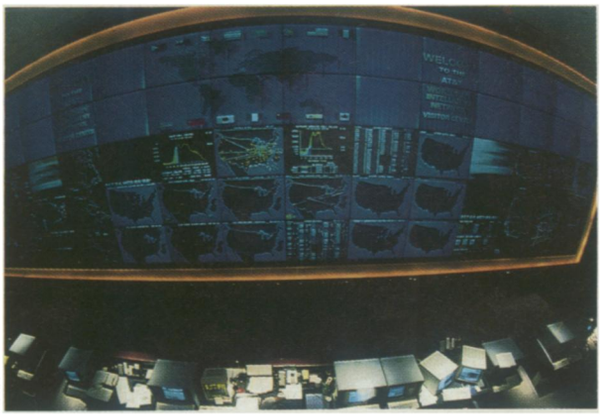
Here's where the fatal software flaw surfaced. Because the receiving switch's communication software was not yet finished with the information from the first call, it had to shunt the second message aside. Because of the programming error, the switch's processor mistakenly dumped the data from the second message into a section of its memory already storing information crucial for the functioning of the communications link. The switch detected the damage and promptly activated a backup link, allowing time for the original communication link to reset itself.

Unfortunately, another pair of closely spaced calls put the backup processor out of commission, and the entire switch shut down temporarily. These delays caused further telephone-call backups, and because all the switches had the same software containing the same error, the effect cascaded throughout the system. The instability in the network persisted because of the random nature of the failures and the constant pressure of the traffic load within the network.

Although the software changes introduced the month before had been rigorously tested in the laboratory, no one anticipated the precise combination and pace of events that would lead to the network's near-collapse.

In their public report, members of the team from AT&T Bell Laboratories who investigated the incident state: "We believe the software design, development and testing processes we used are based on solid, quality foundations. All future releases of software will continue to be rigorously tested. We will use the experience we've gained through this problem to further improve our procedures."

In spite of such optimism, however, "there is still a long way to go in attaining dependable distributed control," warns Peter G. Neumann, a computer scientist with SRI International in Menlo Park, Calif. "Similar problems can be expected to recur, even when the greatest pains are taken to avoid them."



AT&T

On a typical business day, AT&T's sophisticated network-management center in Bedminster, N.J., monitors the handling of 115 million telephone calls. The center's 75-screen video wall depicts the network's performance worldwide.

Even a relatively short, simple computer program can prove difficult to check out, as illustrated by the tremendous effort required to ensure the correctness of the software for the Darlington power station.

Darlington's two shutdown systems operate independently, each using different sensors, different shutdown mechanisms and different computers controlled by software written by separate teams. Their sole purpose is to shut the plant down if the values of certain variables exceed preset limits.

Although shutdown systems have a simple task, the computer-based version designed by Ontario Hydro engineers turned out to be significantly more complex than the straightforward, easily inspected mechanical controls it replaced. Complicated pathways and shared data took the place of individual, obviously connected devices.

Officials at the Atomic Energy Control Board (AECB) in Ottawa, Ontario, which regulates and licenses Canadian nuclear power plants, decided they needed outside help in evaluating the software instructions. "There's only so much you can do by reading it line by line – the usual approach," says AECB's G.J.K. Asmis.

To dig deeper into the reactors' software, AECB turned to Parnas. Known as an outspoken critic of the Strategic Defense Initiative because of its unprecedented reliance on software, Parnas has long argued that computer programmers must take a more disciplined approach to writing software in order to improve its quality and avoid serious flaws. During the 1980s, he and his associates had developed a bank of mathematical techniques for evaluating computer programs.

"When I looked at the code [lines of instructions], it became clear that I

couldn't say if it was okay or not," Parnas recalls. "All I could say was that the documentation [explaining the function of each part of the program] was too vague."

For example, consider the specification: "Shut off the pumps if the water level remains above 100 meters for more than 4 seconds." The sentence appears clear — but what if the water level varied during the 4-second period?

Parnas came up with three different interpretations of this statement, based on different ways of finding the average water level. A programmer could choose only one of the three. Which was correct?

When Parnas checked with the engineers at Ontario Hydro, he discovered that their interpretation, based on long experience with the design of shutdown systems, differed from the three choices he had suggested. This example told Parnas that the specifications for the shutdown software had to be expressed much more precisely.

The engineers proved reluctant to spend additional time writing more detailed specifications. "We argued that even if the specification wasn't written down in a mathematically precise way, an experienced designer would know what it means," Archinoff says.

However, AECB officials were sufficiently concerned about potential problems that they insisted on a thorough review incorporating a variety of software-checking techniques.

The Ontario Hydro team had already systematically subjected their software to a large number of carefully constructed tests designed to ensure that it functioned properly under a variety of circumstances. But planned tests such as these cover only a fraction of the possible paths through the software, and they often miss subtle cases.

Parnas recommended that Ontario Hydro also try random testing — for example, by furnishing to the shutdown systems randomly generated sensor data to see how they responded. "That's often more effective than carefully controlled testing," he says.

Furthermore, because system designers usually can't guarantee that their specifications cover every possible way in which a system will be used, many now perform a hazard analysis. The idea is to consider all the ways in which a system can fail, and then to work backwards through the hardware and software components to determine what factors could cause such failures. This enables designers to incorporate safeguards that specifically prevent these problems from occurring.

"You have to build safety into software," says Nancy G. Leveson of the University of California, Irvine, who pioneered hazard analysis for software. "Just trying to get it correct isn't enough."

The final stage, involving techniques developed by Parnas and his colleagues to prove mathematically that the software does what the requirements ask, proved both exhausting and exhaustive.

Three separate teams went to work. One examined just the computer program and painstakingly determined what each section of the program actually did. A second team converted the systems' original specifications into precise mathematical statements, written out in the form of tables. Finally, a third team tried to find any mismatch between the mathematically expressed specifications and the program functions determined by the first group, and listed all discrepancies.

"None of the jobs was fun, but they were doable," Parnas says. "The effect . . . was to reduce the extremely complex task of reviewing the system to a large number of relatively simple tasks. The tasks were often dull and tiresome, but the systematic procedure . . . made it possible to take breaks and to rotate personnel to prevent burnout."

"They ended up with hundreds of discrepancies, but most were benign," Asmis says. In many cases, reviewers found that the programmers had inserted extra instructions, such as additional safety checks, which were not called for in the specifications. The teams also uncovered a handful of errors. None of the errors proved serious enough to delay or prevent an emergency reactor shutdown.

In the end, despite many minor changes, the two computer programs remained essentially the same as before. "The engineers had put a lot of effort into trying to get it right, and basically they had succeeded," Asmis says.

The entire checkout took about three years. "The checking process had value," Archinoff says. "The problem is that it was extremely costly — very labor-intensive and time-consuming. In fact, if we had to do it again, using the same methods, we wouldn't use software. We'd go back to using hardware."

Many of the frustrations in the checking process could have been avoided if the software designers had written the programs with review in mind. Ontario Hydro engineers and experts from Atomic Energy of Canada, Ltd. — designers of the type of nuclear reactor used in Canada — are now working with AECB to establish standards for future software projects. Then Ontario Hydro personnel will rewrite the Darlington shutdown software to reflect the new requirements.

"You want a program that not only works but also can be understood by more than one or two people," Asmis says.

The Darlington experience with safety-critical software is not yet common in the nuclear industry. In the United States, most existing nuclear power plants use computers only for functions unrelated to plant safety. However, officials at the Nuclear Regulatory Commission in Washington, D.C., believe that software control will inevitably creep into nuclear plant designs, and they are starting to prepare for the task of software evaluation.

Most computer programs don't go through the kind of careful programming and intense scrutiny applied to the Darlington shutdown system or the AT&T telephone network. The process is both costly and time-consuming, and many programmers lack the expertise to use the sophisticated methods necessary for ensuring software reliability.

"Education is important," Stone says. "There are a lot of techniques [for developing reliable software] on the table that are proven and work well, but they still aren't universally practiced."

Moreover, anyone can call themselves a computer programmer and market a software product. "There's no other technology that we depend on to the extent that we depend on software technology that is so unregulated," says software developer John Shore, president of Entropic Research Laboratory, Inc., in Washington, D.C.

It's not surprising, then, that computer programs contain errors and computer systems unexpectedly fail. Often, developers of commercial software work under so much pressure to deliver a product that new programs go out riddled with flaws. "Whether you're a small company struggling to survive or a big company with a big budget, the pressures become enormous, and you end up feeling that you've got to get something out the door to keep the customers satisfied or just to survive," Shore says. "One of the things that saves us is that a lot of customers have come to expect this. They understand how complicated software is."

Indeed, commercial software producers sometimes appear to rely on their customers to do a significant part of the software testing for them. Any user of such software must watch closely for problems and anticipate the possibility of sudden, inexplicable failures.

As computer programs grow larger and more complex, and computer systems keep taking on greater responsibilities, managing the software monster becomes increasingly difficult.

"The problem is intrinsically unsolvable, but you can always do better," Neumann says. "It's a question of system design, of experience, of good software engineering techniques, of recognizing risks, and of continually adapting to a changing environment. There are no easy answers." □