# WARNING: THIS SOFTWARE MAY BE UNSAFE

## Software engineers are looking for ways to eliminate computer program errors that could lead to catastrophe

By IVARS PETERSON

At first glance, the problem seemed minor. The Bank of New York's computer program for handling sales of government securities was supposedly designed to cope with up to 36,000 different issues. But a burst of activity on one day last November uncovered a software glitch. Too little computer storage space had been allocated for keeping track of that many securities.

This error sparked a series of computer problems that corrupted thousands of transaction records. Because the computer couldn't promptly pass on transaction data to the New York Federal Reserve Bank, the Bank of New York ended up owing as much as $32 billion. At the end of the day, it was forced to borrow about $24 billion – pledging all of its assets – to balance its accounts overnight.

This incident, which took two days to clear up, cost the Bank of New York millions of dollars in interest payments. The securities market was disrupted, and some investors lost money because of processing delays. But its greatest impact may have been on federal banking officials worried about the fragility of computer systems that handle financial transactions.

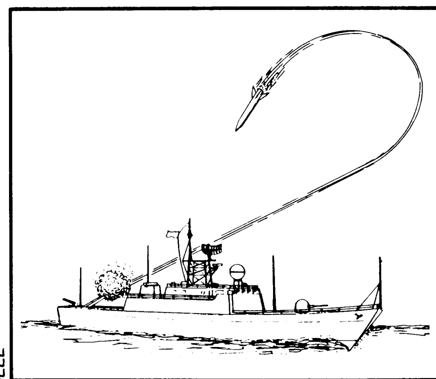
While the Bank of New York case was the most serious problem yet caused by a banking computer malfunction, less serious disruptions occur frequently. In 1985, software and equipment problems delayed end-of-day account settlements more than 70 times, says E. Gerald Corrigan president of the New York Federal Reserve Bank. "It is unrealistic to expect that we will ever achieve a fail-safe payments system," he says. Yet without computers, the system couldn't work as efficiently and economically as it does.

"Like it or not," says Paul A. Volcker, Federal Reserve Board chairman, "computers and their software systems – with the possibility of mechanical or human failure – are an integral part of the payments mechanism. The scale and speed of transactions permit no other approach." Volcker and Corrigan made their remarks last December to a House banking subcommittee investigating the computer failure.

The U.S. banking system is not the only place where a computer failure could have disastrous consequences. Air traffic control systems, nuclear reactors and chemical plants, medical technology, defense and aerospace systems all depend heavily on computer software to function properly. The software reliability issue has also starred in the debate over the feasibility of the proposed Strategic Defense Initiative (SN:7/14/84,p.26).

"Our health and welfare as individuals and as a nation are increasingly dependent on the proper functioning of computer systems," says Herman O. Lubbes Jr. of the Navy's Space and Naval Warfare Systems Command in Washington, D.C. "We have spent little time or effort to assure ourselves that we are taking proper care in the computer's use." Lubbes organized and chaired a recent conference on "Computer Assurance" held in Washington, D.C.

"How can we be sure," asks Lubbes, "that vague or incomplete specifications, design flaws or implementation errors won't result in systems [that] fail to perform when needed or perform incorrectly, . . . threatening human life and welfare?"



IEEE

*The logo for a recent software safety conference shows one possible consequence of a software error.*

The problem is that errors in large, complicated computer programs are practically inevitable (SN:5/14/83,p.312). John Shore, a research scientist at the Naval Research Laboratory (NRL) in Washington, D.C., writes in *The Sachertorte Algorithm* (Viking, 1985): ". . . the typical large computer program is considerably more likely to have a major, crash-resulting flaw than is the typical car, airplane, or elevator. The computer may be the ultimate machine, but today it's less trustworthy than many of its predecessors."

When designing systems and writing



Science Service, Inc. is collaborating with JSTOR to digitize, preserve, and extend access to *Science News*. JSTOR www.jstor.org

computer software, says David L. Parnas, a computer scientist at NRL and Queen's University in Kingston, Ontario, "we need to know what we can and can't do."

Software engineers try to measure the quality of software in several ways. A computer program is considered "correct" if it always does exactly what it's supposed to do; that is, it meets specifications. Except in simple cases and by using, for instance, formal mathematical proofs, such perfection is practically impossible to achieve. Furthermore, although extensive testing may uncover errors, it doesn't guarantee that all of them have been found. And the initial specifications may themselves be incomplete, poorly stated or wrong.

"Reliability" expresses the probability of an error cropping up when a program is in use. High reliability means that the program is unlikely to contain many errors. Again, testing, simulations and long experience with a program help weed out defects. Nevertheless, there is always a chance that something won't work properly.

"We often accept systems that are unreliable," says Parnas. But they should be used only if no possible system failure can lead to a disaster. A writer, for instance, could occasionally afford to lose an article because of a defect in a word processing program. It would be inconvenient but rarely, if ever, life-threatening. The same could not be said for software that controls the firing of a ballistic missile.

Given that perfection is impossible to achieve, measures of "trustworthiness" try to distinguish between errors that are minor and those that could lead to catastrophic results. The idea is to find and eliminate the errors that could cause the greatest harm. In this way, "debugging" efforts are concentrated where they are most needed.

"Even if all failures cannot be prevented," says computer scientist Nancy G. Leveson of the University of California at Irvine, "it may be possible to ensure that the failures that do occur are of minor consequence."

But it's still a formidable task. Errors abound.

Several years ago, an error in the avionics software for the F-16 jet fighter instructed the plane to flip upside down whenever it crossed the equator. Luckily, a simulation uncovered the problem before it happened to a pilot on a mission.

More recently, an F-18's computer, which controlled the firing of a wing-mounted missile, performed the functions it was supposed to – opening the missile rack, firing the missile and then closing the rack. Unfortunately, the rack was closed too soon, and the pilot had to contend with a "live" missile attached to his aircraft. The plane dropped about 20,000 feet before the pilot regained control.

In separate incidents, software bugs were responsible for the death of one patient and injuries to two others when an irradiation unit for cancer therapy generated "inappropriate" doses.

An error in software for evaluating the effects of earthquakes on nuclear reactors forced the temporary closing of five power plants that had been constructed according to the original, flawed computer model.

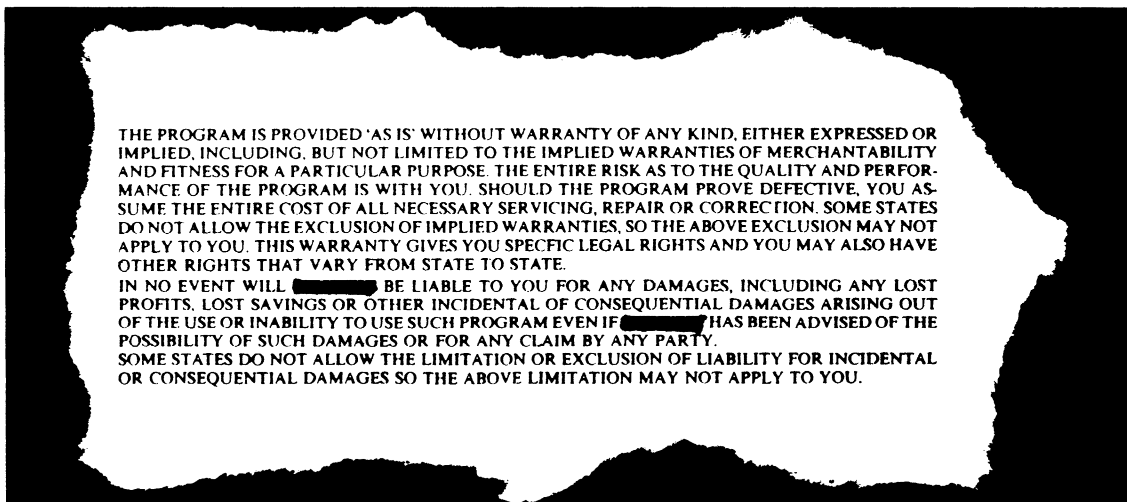
And the list goes on and on. "Experience shows that even the most carefully designed systems may have serious flaws," says Peter G. Neumann of SRI International in Menlo Park, Calif. "In addition, human misuse can compromise even the best systems. We must be extremely careful in doing anything at all." Neumann has documented many "computer-related disasters and other egregious horrors," as he puts it, including the ones described above, in quarterly issues of SOFTWARE ENGINEERING NOTES, published by the Association for Computing Machinery in New York City.

What can be done?

"One option is not to build these systems or not to use computers to control them," suggests Leveson. Alternatives should be looked at. Designs should include backup systems. And a given system shouldn't rely entirely on software to control critical functions.

"For the most part, however, this option is unrealistic," she admits. "There are too many good reasons why computers should be used and too few alternatives."

The experimental X-29 aircraft, for example, with its novel swept-forward wing design, is aerodynamically unstable (SN:1/8/83,p.26). Only a computer can make adjustments quickly enough to fly the airplane. This new jet fighter is now being tested – very cautiously – and software errors are showing up.



THE PROGRAM IS PROVIDED 'AS IS' WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS AND YOU MAY ALSO HAVE OTHER RIGHTS THAT VARY FROM STATE TO STATE.

IN NO EVENT WILL █████ BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OF CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE SUCH PROGRAM EVEN IF █████ HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR FOR ANY CLAIM BY ANY PARTY.

SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

*Few companies that develop and sell computer software guarantee that their products will work without a hitch. Most include a disclaimer, like the one shown, to warn buyers that the software company is not responsible for any problems that a user may encounter because of errors in a program.*

Leveson herself is developing a set of techniques and tools that can be used to enhance software "safety." These attempt to eliminate errors that could lead to failures resulting in death, injury, environmental harm, or loss of equipment and property.

One key element involves imagining the worst possible consequences of a system failure, then working backward to ensure that no possible path through the software leads to those results. Another approach is to examine carefully how well a software package fits in with the rest of the system it controls. Many, if not most, serious accidents are caused by multiple failures, says Leveson, and by complex, unplanned and unfortunate interactions between components of a system.

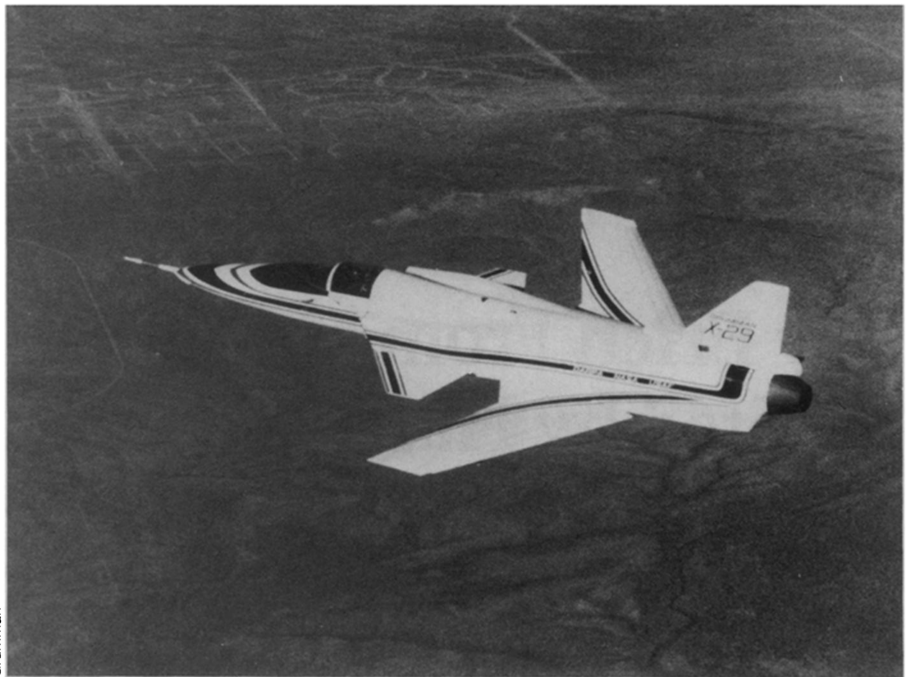
In one famous example, the first space shuttle flight was delayed because two different computer programs, both designed to ensure that the Orbiter could be flown reliably, were not synchronized. Considered separately, the programs worked, but an interface bug prevented the backup flight software from starting up at the right time.

Donald I. Good of the University of Texas at Austin argues that it may be possible mathematically to prove that computer programs – or at least significant parts of them – are completely free of

logical errors. But this would be feasible only if the time it takes, for instance, to prove that a piece of software performs exactly as required for every possible input is short. That means developing automatic or mechanical theorem-provers so that much less human labor is required.

Most of the elements that would go into such methods are now available, says Good. Among them is the use of "functional" programming languages, which make a logical and mathematical approach easier to implement (SN:9/24/83,p.202).

Good and his colleagues have developed several schemes for mechanizing logic. These have already successfully proved the correctness of some simple systems, including one 4,211-line program. "We believe that it is quite possible that the final technology will be able to produce proved systems with less human labor than is required to produce comparable, unproved systems today," says Good.



Grumman

*The X-29 is one example of an aircraft that cannot be flown without a computer. A computer failure here would probably be disastrous.*

Testing of some kind, however, would still be needed. What's missing, says Good, is a basis for formulating precise statements of what the programmer is trying to do in the first place. Too often, software engineers start writing computer "code" before they really understand the problem they are trying to solve.

"We begin building systems without really understanding what we're building," says Lubbes. "We do a very poor job of expressing our intentions."

For some software developers and users, the magic words are now "artificial intelligence" or "expert systems." They suggest that computer programs that attempt to mimic the way human experts analyze and solve problems would also be useful for ensuring that programs are written correctly or for testing completed software.

The difficulty, says Shore, is that expert systems are themselves computer programs, which may contain flaws. In fact, trial-and-error methods now used to build many such systems are extremely difficult to analyze. Adding a new rule, for instance, can have quite unpredictable effects on the way the program responds. Often, the "correct" behavior itself is a matter of opinion.

"While expert systems may be more glamorous than other software," says Shore, "they are not more reliable. Like other large computer programs, expert systems suffer from the inadequacy of testing, the curse of flexibility, the existence of invisible interfaces and other problems." The best systems, he says, are based on precise mathematical methods that impose consistency and logic.

Good expert systems may turn out to be helpful for special software-writing purposes. One such system could be used as an assistant to make sure that specifications are complete. Another could generate suitable tests. Other expert systems could perform the inspections required by the tests — a tedious, error-filled task for humans.

"But there is no guarantee of total coverage," says Stan Letovsky of Yale University. "It just provides a systematic approach."

"Human fallibility is a given," says Shore. "We must recognize this in software engineering." This means writing computer programs more carefully so that they are easier to check and easier to alter when necessary. It means testing programs again and again in many different ways. It means using the programs and watching for flaws.

Unfortunately, this cautious approach is often overlooked in the rush to bring software products to market or by low-bid contractors cutting corners to meet a deadline or to keep costs down in military projects. Moreover, although there are many computer programmers, too few have the skills or knowledge to develop truly reliable and trusted software. Perhaps, says Parnas, people who call themselves software engineers ought to meet some kind of standard.

"There are a lot of good ideas around," says Lubbes. "But research and practice are still far apart."

Furthermore, the need to ensure that software is safe is growing steadily. The Food and Drug Administration (FDA) is searching for a way to inspect and regulate medical instruments that are run by computer programs. "We want safety built into consumer products," says FDA's M. Frank Houston. "But checking for safety must be made less expensive."

NASA faces a massive task in preparing to resume space shuttle flights. Somehow, NASA engineers must check millions of lines of computer software to make sure that there is no "O-ring problem smoldering in that mountain of software," says William M. Wilson, NASA's chief engineer.

No one even knows how much shuttle software is in use altogether. The problem is made more difficult, says Wilson, because much of the software was hurriedly patched together to meet deadlines. It was not designed for ease of testing. "We may also end up with new software if the shuttle is modified significantly," says Wilson. "That would have to be certified too."

What about the future? The next decade may see machines with a million or more processors working in parallel instead of the single processor that sits at the heart of most of today's computers. Individual computer chips may bear as many as a billion electronic devices. Light pulses rather than electronic signals may carry messages. "These new technologies present new problems," says Alfred W. Friend of the Space and Naval Warfare Systems Command. The task of ensuring safety will not get any easier.

"The good news is that computer system technology is advancing," says Neumann. "Given well-defined and reasonably modest requirements, good people, adequate resources and suitably reliable hardware, systems can be built that satisfy their requirements most of the time.

"The bad news," he says, "is that completely guaranteed behavior is intrinsically impossible to achieve."

He concludes that ". . . even if we are extremely cautious and lucky, we must anticipate the occurrence of serious catastrophes in the future." □