Computer System Reliability and Nuclear War

Alan Borning

Associate Professor of Computer Science, University of Washington, Seattle. Dr. Borning's work with Computer Professionals for Social Responsibility resulted in a definitive analysis of the role of computer failure in accidental nuclear war.

False Alerts

On Tuesday, June 3, 1980, at 1:26 a.m., the display system at the command post of the Strategic Air Command (SAC) near Omaha, Nebraska, indicated that two submarine-launched ballistic missiles (SLBMs) were headed toward the United States. (1) Eighteen seconds later, the system showed an increased number of SLBM launches. SAC personnel called the North American Aerospace Defense Command (NORAD), who stated that they had no indication of attack.

After a brief period, the SAC screens cleared. But, shortly thereafter, the warning display at SAC indicated that Soviet ICBMs had been launched toward the United States. Then the display at the National Military Command Center in the Pentagon showed that SLBMs had been launched. The SAC duty controller directed all alert crews to move to their B-52 bombers and to start their engines, so that the planes could take off quickly and not be destroyed on the ground by a nuclear attack. Land-based missile crews were put on a higher state of alert, and battle-control aircraft prepared for flight. In Hawaii, the airborne command post of the Pacific Command took off, ready to pass messages to US warships if necessary.

Fortunately, there were a number of factors which made those involved in the assessment doubt that an actual attack was underway. Three minutes and twelve seconds into the alert, it was canceled. It was a false alert.

NORAD left the system in the same configuration in the hope that the error would repeat itself. The mistake recurred three days later, on June 6 at 3:38 p.m., with SAC again receiving indications of an ICBM attack. Again, SAC crews were sent to their aircraft and ordered to start their engines.

The cause of these incidents was eventually traced to the failure of a single integrated circuit chip in a computer which was part of a communication system. To ensure that the communication system was working, it was constantly tested by sending filler messages which had the same form as attack messages, but with a zero filled in for the number of missiles detected. When the chip failed, the system started filling in random numbers for the "missiles detected" field. (1)

The Question

Due to the short warning times involved - measured at best in minutes – today's nuclear forces could not function without high-speed computers to automate the warning process, control communications, and, should it be deemed necessary, guide missiles to their targets. How reliable are the computers used in the command and control of nuclear weapons? Can they be made adequately reliable? These are the questions addressed in this paper.

The concept of "reliability" extends beyond merely keeping a system running. It invades the realm of system intention or even of what we should have intended, had we only known. To what extent are we able to state and codify our intentions in computer systems so that all circumstances are covered?

"The SAC duty controller directed all alert crews to move their B-52 bombers and to start their engines … Three minutes and twelve seconds into the alert, it was cancelled."

Is it responsible for the USSR or the US to adopt policies which could result in an accidental nuclear war, should a computer system fail? As outlined below, I argue that it is not. The standard of reliability required of military computer systems whose failure could precipitate a thermonuclear war must be higher than that of any other computer system, since the magnitude of possible disaster is so great.

Sources of Failures

Computer systems can fail because of incorrect or incomplete syst-em specifications, hardware failure, hardware design errors, software coding errors, software design errors, and human error such as incor-rect equipment operation or maintenance. Particularly with complex, normally highly reliable systems, a failure may be caused by some unusual combination of problems from several of these categories.

Hardware failures are perhaps the most familiar cause of system failures, as in the June 1980 NORAD false alerts. Individual components can be made very reliable by strict quality control and testing, but in a large system it is unreasonable to expect that no component will ever fail, and other techniques that allow for individual component failures must be used. However, when one builds very complex systems – and a command and control system in its entirety is certainly an exam-ple of a complex system – one becomes less certain that one has antici-pated all the possible failure modes, that all the assumptions about independence are correct. (2, 3, 4) A serious complicating factor is that the redundancy techniques that allow for individual component failures themselves add additional complexity and possible sources of error to the system.

Another potential cause of failure is a hardware design error. Again, the main source of problems is not the operation of the system under the usual, expected set of events, but its operation when unexpected events occur. For example, timing problems due to an unanticipated set of asynchronous events that seldom occur are particularly hard to find.

> *"We can have confidence in complex systems only after they have been tested for a considerable time under conditions of actual use … The untstability of the [nuclear] warning and control system under highly stressed conditions is grounds for considerable concern."*

It is in the nature of computer systems that much of the system design is embodied in the computer's software. The cost and complexity of the software typically dominate that of the hardware. It is generally accepted that reliability cannot be "tested into" a software system; it is necessary to plan for reliability at all points in the devel-opment process. As with high-reliability hardware, there are codified standards for how critical software is to be specified, designed, written, and tested. Even so, errors may be introduced at any of the steps in software production: requirements specification, design, implementation, testing and debugging, or maintenance. (5, 6, 7)

Errors in the system requirements specification, for both hardware and software, are perhaps the most pernicious. We must anticipate all the circumstances under which the system might be used and describe what action it should take in each situation. For a complex system, one cannot foresee all of these circumstances. We can have confidence in complex systems only after they have been tested for a considerable time under conditions of actual use. Short of having many periods of great international tension and high military alert – clearly an unacceptably dangerous proposition – the nuclear weapons command and control systems cannot be tested under conditions of actual use. Testing under the most extreme conditions in which these systems are expected to function – that of limited or protracted nuclear war – is an impossibility. The untestability of the warning and control systems under highly stressed conditions is grounds for considerable concern.

Errors may also be introduced when the requirements are translated into a system design, as well as when the design is translated into an actual computer program. Again, the sheer complexity of the system is a basic cause of problems. Anyone who has worked on a large computer system knows how difficult it is to manage the development process. Usually, no one person understands the entire system completely.

Program maintenance, either to fix bugs or to satisfy new system requirements, has itself a high probability (typically from 20 to 50 percent) of introducing a new error into the program.

Another source of failure is human operator error. People do make mistakes, despite elaborate training and precautions, especially in time of stress and crisis. On November 9, 1979, a test tape containing simulated attack data, used to test the missile warning system, was fed into a NORAD computer, which through human error was connected to the operational missile alert system. During the ensuing six-minute alert, ten tactical fighter aircraft were launched from bases in the northern United States and Canada. (1)

"On October 5, 1960, the warning system at NORAD indicated that the United States was under massive attack by Soviet missiles whith a certainty of 99.9 percent. It … had spotted the rising moon."

Human error becomes more likely under the influence of alcohol or drugs. Dumas cites some worrying statistics about alcohol, drug abuse, and aberrant behavior among American military personnel with access to nuclear weapons. (8) Alcoholism is a health problem in the Soviet Union and may be a problem among Soviet military personnel as well.

Some Instructive Failures

It is instructive to look at a few of the impressive failures of systems designed to be highly reliable. Most examples concern US systems, since this is the data available to the author. One would expect similar failures in the USSR or any other industrialized nation.

The June 1980 NORAD false alert described in the opening of this paper is an example of a hardware failure. However, this false alert also illustrates hardware design error. It was a grave oversight that such critical data, reporting a nuclear attack, was sent without using standard, well-known error-detection techniques. (1)

"Incidents such as Three Mile Island and Chernobyl, the tragic explosion of the space shuttle Challenger in 1986, and the 1965 Northeast power blackout are sobering reminders of the limitations of technology."

Another example of hardware failure was the total collapse of a Department of Defense computer communications network in October 1980. This failure was due to an unusual hardware malfunction that caused a high-priority process to run wild and devour resources needed by other processes. This communications network was designed to be highly available - the intent being that it should prevent a single hardware malfunction from being able to bring down the whole network. It was only after several years of operation that this problem manifested itself.

The launch of the first space shuttle was delayed at the last minute by a software problem. For reliability, the shuttle used four redundant primary avionics computers, each running the same software, along with a fifth backup computer running a different system. A patch to correct a previous timing bug created a 1 in 67 chance that, when the system was turned on, the computers would not be properly synchronized. There are a number of noteworthy features of this incident. First, despite great attention to reliability in the shuttle avionics, there was still a software failure. Second, this failure arose from the additional complexity introduced by redundancy in an attempt to achieve reliability. And third, the bug was introduced during maintenance to fix a previous problem.

There are many examples of errors arising from incorrect or incomplete specifications. On October 5, 1960, the warning system at NORAD indicated that the United States was under massive attack by Soviet missiles with a certainty of 99.9 percent. It turned out that the Ballistic Missile Early Warning System radar in Thule, Greenland, had spotted the rising moon. Nobody had thought about the moon when specifying how the system

should act. Gemini V splashed down one hundred miles from its intended landing point because a programmer had implicitly ignored the motion of the earth around the sun - in other words, he had used an incorrect model. In 1979, five nuclear reactors were shut down after the discovery of an error in the program used to predict how well the reactors would survive in earthquakes. One subroutine, instead of taking the sum of the absolute values of a set of numbers, took their arithmetic sum instead.

In hindsight, the blame for each of the above incidents can be assigned to individual component failures, faulty design, or specific human errors, as is almost always the case with such incidents. But the real culprit is simply the complexity of the systems, and our inability to anticipate and plan for all of the things that can go wrong.

What about similar failures in the Soviet warning systems? I have been unable to ascertain whether or not such failures have occurred, and to date the Soviet government has not revealed them if they existed. However, the Korean Airlines Flight 007 incident, in which a civilian aircraft was shot down by the Soviet Union more than two hours after it had entered Soviet airspace and just before it was back over international waters, would seem to indicate that the Soviet command and control system has problems. The fatality rates for American astronauts and Soviet cosmonauts and the nuclear power plant failures at Three Mile Island and Chernobyl also indicate comparable failure rates of high reliability systems in both countries.

Incidents such as Three Mile Island (7) and Chernobyl, the tragic explosion of the space shuttle Challenger in 1986, and the 1965 Northeast power blackout are sobering reminders of the limitations of technology.

Prospects for Future Improvements

What are the prospects for improving the reliability of military computer systems in the future? Substantial progress is possible simply by using state-of-the-art hardware and software engineering techniques. (5, 6, 9) A system, like NORAD's, that in 1980 used 1960s vintage computers or transmitted critical data without error detection is not state-of-the-art.

State-of-the-art techniques can help, but what are the practical and theoretical limits of reliability, now and in the next decade? The Department of Defense is engaged in several efforts to develop new technology for software production and to make it widely available to military contractors. The Software Technology for Adaptable, Reliable Systems program, and the Software Engineering Institute at Carnegie-Mellon University are examples. Use of these techniques should decrease, but not eliminate, errors in moving from the specification to the program.

In the long term, formal techniques such as proofs of program correctness (program verification), automatic programming, and proofs of design consistency have been advocated as tools for improving computer system reliability. (5) In a proof of program correctness, either a human or a computer proves mathematically that a program meets a formal specification of what it should do. In automatic programming, the program is written automatically from the specification. In a proof of design consistency, the proof must show that a formal specification satisfies a set of requirements, for example, for security or fault tolerance.

But program verification and automatic programming techniques can offer no help with the hardest and most intractable problem in the construction of software for complex tasks, such as command and control systems: specifying what the system should do. How does one know that the specification itself is correct, that it describes what one intends? Are there events that may occur that were simply not anticipated when the specification was written?

"Both the practical and theoretical limits of reliability bumb up against this problem of specification. It constitutes the major long-term practical barrier to constructing reliable complex systems."

A proof of correctness, for example, only shows that one formal description (the specification) is equivalent to another formal description (the program). It does not say that the specification meets the perhaps unarticulated desires of the user, nor does it say anything about how well the system will perform in situations never imagined when the specification was written.

For example, in the 1960 false alert, proving that the system met its specifications would not have helped since no one thought about the rising moon when writing the specifications. The term "proof of correctness" is thus a misnomer - a better term might be "proof of relative consistency."

Both the practical and theoretical limits of reliability bump up against this problem of specification. It constitutes the major long-term practical barrier to constructing reliable complex systems. The answers to such critical questions as, "Will the system do what we reasonably expect it to do?" or "Are there external events that we just didn't think of?" lie inherently outside the realm of formal systems. Computer systems (including current artificial intelligence systems) are notoriously lacking in common sense: The system itself will typically not indicate that something has gone amiss and that the limits of its capabilities have been exceeded.

Conclusions

How much reliance is it safe to place on life-critical computer systems, in particular, on nuclear weapons command and control systems? At present, a nuclear war caused by an isolated computer or operator error is probably not a significant risk, at least in comparison with other dangers. The most significant risk of nuclear war at present seems to come from the possibility of a combination of such events as international crises, mutually reinforcing alerts, computer system misdesign, computer failure, or human error.

A continuing trend in the arms race has been the deployment of missiles with greater and greater accuracies. This trend is creating increasing pressure to consider a launch-on-warning strategy. Such a strategy would leave very little time to evaluate the warning and determine whether it was real or due to a computer or human error. We would be forced to put still greater reliance on the correct operation of the warning and command systems of the US and the USSR. Deployment of very accurate missiles close to an opponent's territory exacerbates the problem.

More exotic weapons systems, such as envisioned in the Strategic Defense Initiative, equipped with extremely fast computers and using artificial intelligence techniques may result in battles (including nuclear ones) that must be largely controlled by computer. (9)

Where then does that leave us? There is clearly room for technical improvements in nuclear weapons computer systems. I have argued, however, that adding more and more such improvements cannot ensure that they will always function correctly. The problems are fundamental ones due to untestability, limits of human decision making during high tension and crisis, and our inability to think through all the things that might happen in a complex and unfamiliar situation. We must recognize the limits of technology. The threat of nuclear war is a political problem, and it is in the political, human realm that solutions must be sought.

References

1. Barry Goldwater and Gary Hart, *Recent False Alerts from the Nation's Missile Attack Warning System*, Report to the Committee on Armed Services, United States Senate (Washington, D.C.: Government Printing Office, 1980).

2. William M. Arkin, "Nuclear Weapon Command, Control, and Communications," in *World Armaments and Disarmament: SIPRI Yearbook 1984* (London and Philadelphia: Taylor & Francis, 1984), pp. 455-516.

3. Bruce G. Blair, *Strategic Command and Control* (Washington, D.C.: Brookings Institution, 1985).

4. Paul Bracken, *The Command and Control of Nuclear Force* (New Haven: Yale University Press, 1983).

5. W. Richards Adrion, Martha A. Branstad, and John C. Cherniavsky, "Validation, Verification, and Testing of Computer Software," *ACM Computing Surveys*, Vol. 14 No. 2 (June 1982), pp. 159-192.

6. N. G. Leveson, "Software Safety: Why, What, and How," *ACM Computing Surveys*, Vol. 18 No. 2 (June 1986), pp. 125-163.

7. Charles C. Perrow, *Normal Accidents: Living with High Risk Technologies* (New York: Basic Books, 1984).

8. Lloyd J. Dumas, "Human Fallibility and Weapons," *Bulletin of the Atomic Scientists*, Vol. 36 No. 9 (November 1980), pp. 15-20.

9. David L. Parnas, "Software Aspects of Strategic Defense Systems," *American Scientist*, Vol. 73 No. 5 (September-October, 1985), pp. 432-440.