# SOFTWARE FAILURE : HANDLE WITH CARE

**Shamim Akhtar Tushar**

In this increasingly software controlled world, faulty softwares are turning up to be the cause of danger. This is particularly true as these softwares are engaged in translating the flick of a pilot's wrist into the movement of a wing flap, slowing a braking car on a slippery road or switching a super fast train from one busy track to another. Software is becoming integral to the operations of the latest medical devices, communication networks and of course, weapons. And all these are happening even though engineers and progra-mmers fully know that the most thoroughly tested computer software is also susceptible to glitches, viruses and bugs.

Three years ago, Britains nuclear regulatory agency sounded an unusual alarm. In an internal report that was leaked to the press, the agency implied that the Sizewell B nuclear power plant, which was about to begin operating on the Suffolk coast of England, might not be safe. The problem was not one of the usual ones, such as plant's evacuation procedure or the disposal of its hazardous nuclear waste. Rather the danger lay in software designed to manage the reactor in case of an emergency. Precisely, it was the reactor's 'Primary Protection System (PPS)' at fault. PPS is a complex programme of 10,000 lines of computer code designed to shut down the Sizewell B if its temperature begins to rise or any other condition poses a danger. If the PPS were to fail during such an emergency, one of the Europe's most densely populated regions would be at heightened risk of a nuclear melt down. The fear was confirmed when the operator of the plant, Nuclear Electric, ran the PPS through a battery of tests. The results were not comforting, the software failed almost half of them.

This case, though alarming, is not the first in which software has played a suspiciously large role in public safety. In the mid of 1980s a microprocessor-controlled cancer therapy machine called the Therac-25, made by Atomic Energy of Canada, ran amok in several US and Canadian hospitals, shooting over doses of radiation into the bodies of at least 6 patients, killing or seriously injuring them. In its investigations, US. Food and Drug Administration found that in several of cases a certain rarely invoked combination of keystrokes caused the machines software to go weird.

During the gulf war, a software glitch was partly responsible for interfering with a patriot missile's radar and tracking system, according to US. Army investigators. The defective software threw off the missile's timing by one-third of a second, enough for it to miss an Iraqi scud missile that, on February 25, 1991, killed 28 soldiers and wounded 98 others in a barrack in Saudi Arabia.

In another incident in November 1994, a software bug within the Oregon Tel Co. Credit union's new automated-teller-machine network allowed cyberjackars with a stolen credit card to bypass what should have been a daily withdrawal limit of a few hundred dollars. By making bogus deposits totaling some $80,000, they were able to make over 700 with drawls amounting to $ 3,50,000 before they were caught.

Thus, as people who have to use software every minute rather than every hour, are facing a dilemma— the technology is more widely applicable than ever, but it has grown so complex, that there is no way to predict or test how it will perform in the real world. Although a computer programme may indeed have a finite number of lines of code, it is like a flute, whose mere 6 or 8 holes can yield a virtually infinite number of melodies.

Given such complexity, it is not surprising that the list of software failures in safety-critical systems has been growing rapidly in recent years. Software often fails because of faulty logic within the code itself or because of the flaws in the original specifications, which are the technical descriptions of how the software is supposed to behave. According to the software safety experts, the best answer to the software reliability problem lies in a system of programming known as the 'Formal method'. Since software is too complex to test in the real world situation, the only solution is to ensure that the software is designed right in the first place and to prove that it works mathematically.

Formal method begins with using mathematical equations rather than English sentences when specifying what a programme is supposed to do. For example, instead of the chief physician's simply telling a programmer to write a programme that sounds an alarm bell when the diastolic pressure of a patient goes below 50 mm of Hg, a formal specification would lay out perhaps 100 different mathematical equations needed to accomplish the task. By proving that the underlying logic of these equations are sound, a mathematician can show that the software will do what it is supposed to do, even before any code is written. When the programmers have finished writing the actual computer code, a validation programme would make sure that the code does what the equations specify. In this way, formal methods can prove that the software is safe. Another factor that can reduce the risk of software failure is the cautious attention to the human errors, particularly of those who are involved in investigating any such sort of failure.

If failures are given proper priority and dealt with appropriate seriousness, it is likely that such mishaps can be prevented.

References :

1. Software failure & backup : An IBM analysis
2. Reader's Digest : November 1995
3. Discover : May 1996
4. The Futurist : January 1996.

p i n p o i n t y o u r c h o i c e

pentium intel

100MHz, 120MHz, 133MHz

massive® COMPUTERS

PHONE 862856,864058

massive PROFESSIONAL PC COMPUTERS®

we deserve your desire...

95/1 New Elephant Road, Zinnat Mansion, 1st floor, Dhaka 1205 Fax: 88-02-865460 Attn massive