

# PROGRAMMING AND THE PUBLIC TRUST

W. Robert Collins and Keith W. Miller

The public's generosity toward programmers and animosity toward machines are both misplaced: computers do exactly what they are told to do with great reliability.

Programmers must take responsibility for how their programs behave.

Even if a complete and unambiguous description of the desired behavior [of a computer program] were available, practical limitations make it impossible to guarantee correct behavior via testing.

In the absence of exhaustive testing, programmers must try more focused testing, formal analysis techniques, and other indirect methods to catch as many problems as possible before a program is released

For large, complicated programs, programmers do not expect to remove all the errors and strive only to remove as many as possible before schedules and budgets require the programmers to say that is good enough

Computer programs, once installed in a computer, are rigid, formal systems.

As the world and our perceptions of the world change, we biological creatures can adapt; computer programs do not. [\[Recent developments may alter this.\]](#)

When a United States Navy ship shot down an Iranian civilian airliner, the captain and crew apparently misinterpreted data shown on a computer screen. Was the computer program "wrong"? Not in a technical sense. But the program was not as helpful as it could have been in this critical situation.

Reusable software "parts" have gained importance in programming. However, the flexibility of software and the wide range of programming applications have made it more difficult than was anticipated to standardize software parts.

Bridge builders can hedge their bets by "over-engineering": over-engineering is difficult for programmers; extra code does not necessarily mean a "stronger" program. In fact, the longer a program becomes, the more errors it tends to have. Over-engineering for programmers *should* mean using other sorts of [non-software] backup for the software in case their programs fail.

This can mean physical backups (manual control of computer-controlled automobile brakes), physical fail-safe features (physical impossibility of all green lights in computer-controlled traffic lights), computer backups (two separate computers in fly-by-wire F16 aircraft), human monitoring (by nurses of computer-generated pharmaceutical instructions), and so on. Over-engineering for programmers may be more expensive than the costs saved by using computers in the first place.

As programmers and other engineers are becoming more concerned with safety, they are beginning to design computer-controlled systems with mechanical interlocks and backups.

Computer professionals are building enormous suspension-bridge programs to travel over great, dangerous chasms. These bridges are not pretty, and they are not strong enough. Malfunctioning X-ray machines, long-distance telephone disruptions, space-shuttle delays, and multi-million-dollar bank errors all illustrate the dangerous nature of our computer bridges.

The public should demand better quality software where automation is appropriate and should resist automation in riskier applications.

If we builders of programs do not encourage realism about our computer bridges, the public will rightly condemn us when some of our bridges crumble.