



Michael Lutz, Rochester Institute of Technology

COMPLEX SOFTWARE: UNSAFE AT ANY LEVEL

Spanning software's
diverse theories,
practices, and
philosophies.

IT IS NATURAL FOR SOFTWARE DEVELOPERS to shudder when they consider the increasing use of software in mission-critical and safety-critical systems. It's not that software should never be used in such systems, but rather that a golly-gee-whiz optimism seems to pervade many such projects, downplaying the substantial risks involved and emphasizing the manifest advantages. In *Digital Woes*, Lauren Wiener does a service to the profession and to humanity as a whole by clearly discussing — in plain prose — many problems to which software is prey. At the end of the book, even the nontechnical reader will understand the need to be very cautious in applying this technology.

FATAL FAILURES. The first chapter presents "13 tales of woe," ranging from the annoying to the terrifying. Included are well-known incidents, such as the Patriot missile timing error, which may have contributed to the deaths of 28 soldiers in Dhahran during the Gulf War, and the software in the Therac-25 radiation-therapy machine, which has been implicated in at least two deaths. But Wiener also discusses more mundane problems, such as the delays in shipping *On Location*, a text-searching utility for PCs. All this serves to introduce the real issue: Why software has bugs and why they are so hard to eradicate.

The problems inherent in and often unique to software development are discussed in chapters two through four. The primary technical problems are the quantized, discontinuous nature of software and the lack of a direct connection between software and physical reality. The first problem is one that traditional engineering disciplines rarely deal with. Mechanical and electrical system designs depend on the fact that small changes in input will result in small, predictable changes in output. Of course, a "small change" in the digital world is a single bit, but the resulting output can vary wildly. I was personally involved with

a system where a 1-bit error (timing-dependent, of course) changed a communications-release operation into a system-reset operation, wreaking havoc.

The second problem is at least as serious: In creating a software system, we create a model, one that purposefully excludes much real-world richness as irrelevant. The problem arises when the models become so complex we can no longer determine if we've excluded just the right stuff — or if we even *know* what we have excluded. Of necessity, a specification reduces the infinite richness of the world to a finite, approximate model. In so doing, we run the

risk of excluding crucial information for reliable and safe system operation. Writing software to spec is difficult; getting the spec itself right is a Herculean task for systems of even moderate complexity.

Wiener does not overlook more mundane human issues: The over-optimism of both developers and managers; software's siren song of infinite malleability and adaptability; and the pressures that lead to hasty delivery of unreliable software. But her basic tenet is that *any* software product necessarily contains multiple flaws: flaws of implementation, design, and modeling — flaws that eventually trigger faults, often at the least propitious time.

ASSESSING THINGS TO COME. Having properly chastened developers and enlightened the general public, Wiener proceeds to discuss current ambitious projects based on digital technology and incorporating immense amounts of software. While it is obvious that she opposes many of these projects, the discussion never degrades into the strident polemic of a fanatic. Instead, she meticulously details the benefits *and* the risks, and asks that we all consider whether the former compensate for the latter. Perhaps her most valuable contribution is to puncture the pseudostatistics that pervade the descriptions of such sys-

WHY DOES SOFTWARE HAVE BUGS AND WHY ARE THEY SO HARD TO ERADICATE?

Editor: Peter G. Anderson
Rochester Institute of Technology
1 Lomb Memorial Dr.
Rochester, NY 14623-0887
pgo@cs.rit.edu

tems. In many cases, we have *no* idea as to what the risks really are, nor can we develop quantitative models whose results meet widespread agreement.

In the final chapter, Wiener proposes some areas in which digital technology may make a positive contribution. What I found most enlightening, however, was her list of seven questions that should be asked of all software projects (but most especially those that are mission- or safety-critical). I won't repeat the list here. Instead, I recommend that you get the book yourself, read it, then decide

whether these seven questions are necessary and sufficient.

A note in passing: A recent issue of *IEEE Software* on software safety contained an article on safety analysis of pacemaker software ("Retrofitting Software Safety in an Implantable Medical Device," Jan. 1994, pp. 41-50). I was astounded to learn that a modern pacemaker system may contain over 500,000 lines of code (in C!). In addition to stimulating the heart muscle when necessary, these systems let physicians program the pacemaker parameters and

access stored monitoring information. Perhaps this is all for the best, and perhaps these pacemakers are a significant advance over their predecessors. But I'm skeptical — and I'm certain Wiener is as well.

Digital Woes: Why We Should Not Depend on Software by Lauren Ruth Wiener, Addison-Wesley, Reading, Massachusetts, 1993, 245pp.

READER SERVICE 100