

Death by Software

On December 7th, along with hundreds of others at the Paris Orly airport, I spent the day waiting for a flight to London. As we found out after a few hours, all flights to the London area were cancelled because of a communications glitch caused by a software problem in the National Air Traffic Service (NATS) internal phone system.

As the BBC reported, the software failure happened when the controllers working overnight were due to hand over to the controllers on the day shift at around 06:00 GMT. "To be clear, this is a very complex and sophisticated system with more than a million lines of software," the spokesman added. "This is not simply internal telephones, it is the system that controllers use to speak to other [air traffic control] agencies both in the UK and Europe and is the biggest system of its kind in Europe," a NATS spokesperson was reported as saying.

Eurocontrol, which manages European air safety, said around 1,300 flights, or 8% of all air traffic on the continent, had been severely delayed; almost 10% of flights at Heathrow were cancelled.

A gentlemen *en queue* with me turned and said, "Well, at least no one dies from software problems."

Would that this gentlemen were to be correct.

Computers are increasingly being introduced into safety-critical systems and, as a consequence, have been involved in accidents. Two of the most widely cited software related accidents in safety-critical systems involved computerized radiation therapy machines.

The first, involved the Therac-25 radiation machine. Between June 1985 and January 1987, six known accidents caused massive overdoses by the Therac-25, resulting in deaths and serious injuries.

Part of the Therac-25 tragedy was caused by a simple update to the user interface allowing users to edit individual data fields, where before all data fields had to be re-entered if a data entry error occurred. The software did not recognize the changes due to timing problems.

Then again, in January 2006, Lisa Norris (then 15 years old) was a patient at the Beatson Oncology Centre (BOC) in Glasgow, Scotland. While undergoing a series of doses of radiation therapy for a relatively rare and complex brain tumor, it was discovered that the 17 dose fractions received by Ms. Norris were some 58% higher than the

prescribed dose fractions. Ms. Norris died from the over-exposure in October 2006.

Despite what can be learned from such accidents, fears of potential liability or loss of business make it difficult to find out the details behind serious engineering mistakes.

Dr. John Wreathall of the Resilience Engineering Group says, “Placing barriers in the way of widespread dissemination of relevant details of adverse events is a way of preventing learning in any organization. Bear in mind that one hallmark of a resilient organization is that it is prepared not only for its own failures, but those which it can learn from others—the more resilient an organization is, the larger are the lessons it has learned from others.”

This can manifest itself in several ways, including recognizing a broader set of challenges that the organization can face (including those it creates for itself by its own activities), it helps the organization better understand “what went wrong”, it expands the models of failure beyond the simple standard models, and it helps the organization calibrate itself against the experience of others. Of course, just having the data available will not in itself ensure safety. But by cutting off the dissemination of data publicly will ensure that accidents will be repeated.

Furthermore, these problems are not limited to the medical industry. It is still a common belief that any good engineer can build software, regardless of whether he or she is trained in state-of-the-art software-engineering procedures.

As Dr. Nancy Leveson wrote in her Therac-25 investigation report, “Most accidents are system accidents; that is, they stem from complex interactions between various components and activities. To attribute a single cause to an accident is usually a serious mistake. We want to emphasize the complex nature of accidents and the need to investigate all aspects of system development and operation to understand what has happened and to prevent future accidents.”

Software engineering is a young discipline: a liberal estimate puts its age at 63 years old. When civil engineering was the same age, the wedge had not yet been invented. We should not be surprised that software, and its practitioners, have few construction standards, procedures, guilds, review boards, board examinations, licensing standards, acceptable manufacturing practices, or continuing education requirements such as we may find in structural engineering, law, architecture, or even massage therapy.