The Day of the Three Glitches

Electronic Design

Richard Kenner, AdaCore

Tue, 2015-08-25 11:03

Yet again we're confronted with a day of technical "glitches," this time three on one day (July 8). Each was seemingly unrelated, but each caused significant disruptions.

Related

What Does It Mean to Say that Code Must Work All the Time?

Stabilize Software Upgrades in Critical Systems

Enforce Programming Standards to Eliminate Human Error

The least serious was the *Wall Street Journal*, where some problem caused the publication's website (or at least part of it) to go down. We're told not to worry, that they're "working to be back up as quickly as possible," and it indeed came back up within a short amount of time. You could argue that no lasting damage was done here, just a few people who couldn't get their "news fix" immediately.

There was also some sort of "technical issue" at the New York Stock Exchange, which was later described as a "configuration problem" of some sort. It caused one of the longest outages of the NYSE in history. But you could argue that we have enough redundancy, so this didn't cause major problems. Many other exchanges trade NYSE stocks, so people who wanted to buy and sell NYSE stocks still could and prices were still available; therefore, no major loss.

The United Airlines failure of the same day wasn't quite as easy to chalk off as causing no major losses. Again an unspecified issue, this time described as network- or communications-related, prevented airport agents from accessing all or part of their reservation system. This caused United to request the FAA to issue a nationwide ground stop of their flights. However, it also affected flights in the air that were landing, since there were no gates at which to park some of those aircraft.

The resulting delays caused problems much of the day, with many passengers missing their connections. There's no way to know how much loss that caused. Perhaps some people missed weddings or funerals. Perhaps others missed important meetings. There's no way to put a dollar value on it, but there clearly was a major loss.

One wonders whether this was the same sort of "communication problem" that shut down the issuing of United States visas. This problem, described as "hardware," took nearly two weeks to fix and resulted in delays of more than a month to issue some visas. Here, the costs were quite known and largely due to the failure occurring when H-2 visas were needed for agricultural workers. And who knows about the personal costs, of people missing weddings, funerals, business meetings, and so on. AdaCore itself was affected by this—one of our employees got delayed in a move to the U.S.

Because of the heightened attention on "cyber-attacks," the three failures on the same day were brought to the attention of the President, and high government officials called the CEOs of the related companies to assure everybody and each other that this was not caused by an attack. But, as others have also pointed out, this should not be reassuring.

It might have been better if we could blame each of these things on a single, external source. Instead we can "merely" blame it on three separate failures. To me, and many others, this is not "reassuring." Especially in the case of the United "glitch," many have blamed it on the fact that "much of the airline industry's computer and technology programs are old and have been cobbled together." That's undoubtedly true, but misses the point by quite a bit.

pected Failure

More significantly, these failures are caused by a culture where software is expected to fail. The fact that I have to reboot my smartphone every week or so because it seems to get into a mode where it's running very slowly is somehow expected—we don't expect these things to work reliably. We're trained to not have much expectation for most software.

But that's "most," not "all!" We have very strict processes for developing software that runs our airplanes. For instance, no fatalities have ever resulted from defects in that software, so we can reasonably expect that such software won't kill us. However, for some completely unclear reason, we don't have the same expectation for the software that runs our cars or runs the medical devices that assist us or are implanted within our bodies. That's because such software is not subject to similarly strict development processes. In fact, it's not mandated to have any restrictions in the development process at all, at least in the U.S.

Software Matters

The absolute safety record in avionics shows we can do better, and that the often expressed view that there's no way a complex piece of software can ever be reliable is quite false. Obviously, it's not economically practical to spend the same amount of time showing the correctness of a line of software for a game as is it for a line of software that runs an airplane. Nonetheless, it's wrong to conclude that the only software that matters is software capable of killing somebody if it fails. All software matters.

If we're to go to the trouble of writing software, we must believe such software will be valuable. And, with the exception of games and similar entertainment, people will depend on that software to function properly. While it may not kill anybody if it doesn't, it could cost them money or mean they can't go to some important function, such as a wedding. And let's not forget Knight Capital, which lost $460M in 30 minutes back in August of 2012 due to a software "glitch."

fortunately, this isn't something we can fix overnight. There's no "magic bullet." It's not like we don't have quate tools. Although we can always do better, many tools and methodologies have been proven by decades of avionics development to be effective in greatly reducing the number of software failures.

The problem is that we have two different software-development cultures. We have one for the avionics community (and the few others such as some rail controls) in which there's a strong culture of strict methodologies and testing aimed at creating highly reliable software. And then there's the majority of the remaining software community, in which reliability is far less important than pushing out product as fast as possible.

I'm certainly not suggesting it's in society's best interest to greatly increase the cost of creating the next fun game for our phones. However, we do need to take a much more careful look at the processes for developing the software for things we depend on, such as for those phones themselves and for airlines, hotels, transit systems, visa processing, news websites, and many other things in our daily lives. Companies may think they're saving money by not worrying about reliability when developing software, but that's betting the company that there won't be a serious loss due to a software failure. Knight Capital lost that bet.

Richard Kenner is the co-founder of AdaCore.