## Current standards and certification

The NRC report outlines some of the significant best practices and identifies the expertise developers need. The IEC 61508 standard divides safety-related systems into five safety integrity levels (SILs). Table 1 defines these levels for high-demand systems.

The standard recommends the use of particular development processes for each SIL. This makes little sense. There is little published data on the effectiveness of different development processes in creating software with low probabilities of unsafe failure, and it is usually impractical to provide evidence that could properly justify claims even for SIL 1. So, by focusing on probability rather than explicit properties, the standard pushes engineers into aiming for a target they cannot show they have hit. Then, by highly recommending different development methods for each SIL, the standard leads engineers to use a set of methods that might not be the most cost-effective way to make the software and system as safe as possible. The result might be that the existence of IEC 61508 and similar standards in their current form will actually *reduce* safety.

That some excellent professionals have followed such standards and built systems that, with hindsight, had an acceptably low rate of unsafe failures is irrelevant. Such achievements provide no evidence that the standards are either adequate for achieving safe software or cost-effective. Belief in the effectiveness of RTCA DO-178, IEC 61508, and similar standards is superstition, not science or engineering.

## New standardization process

These criticisms apply to most standards for critical software, not just to IEC 61508 and DO-178. When the first standards for safety-related software were written, most engineers had a poor understanding of software failures and systematic failures in general. As a result, the standards were written analogously to hardware standards, with emphasis on failure probabilities, manufacturing processes, and testing. But software differs from hardware: It is extremely complex, and software behavior is generally not continuous.

**Table 1. The IEC 61508 standard's safety integrity levels.**

| Probability of failure per hour | Safety integrity level |
|---|---|
| More than $10^{-5}$ | Unclassified |
| $10^{-5}$ to $10^{-6}$ | SIL 1 |
| $<10^{-6}$ to $10^{-7}$ | SIL 2 |
| $<10^{-7}$ to $10^{-8}$ | SIL 3 |
| $<10^{-8}$ to $<10^{-9}$ | SIL 4 |

The current processes for creating international standards for safety-critical software are based on achieving some consensus among participants. This might work well when the objectives are appropriate and achievable, the industrial processes are mature and effective, and the standard's purpose is to protect customers and the public against developers who do not follow the established best practices. That is not the situation with safety-critical software.

Our targets are unrealistic: There is no science that could justify claims that a newly developed system will fail no more often than once every billion hours, with 99 percent confidence. There is huge economic pressure to use commercial, off-the-shelf software wherever possible, although the vendors of such software rarely, if ever, provide adequate evidence for a credible safety case: Indeed, if we demanded scientifically strong safety cases, it would be necessary to abandon probabilistic failure targets below about $10^{-4}$ per hour. Such honesty is both necessary and desirable to allow a rational discussion about the costs, risks, and benefits of new technological developments.

Our industry develops software using ambiguous design notations and writes programs with inadequately defined languages that make it easy to make mistakes and hard to find them once made. Our software developers quickly adopt new fashions, but only slowly take advantage of real advances in computer science and software engineering—such as deep static analysis, model checking, and SAT solving.

We need software safety standards that let the industry make safety claims for which we can provide credible evidence, that lead the industry to use the best computer science and software engineering, and that stimulate a market for much better design languages, programming languages, and supporting tools.

In short, we need change—which can be uncomfortable and expensive, and let new companies challenge the hegemony of established suppliers. The current standardization committees have demonstrated a serial failure to deliver such changes. We need a fresh approach. ■

*Martyn Thomas is an independent consultant and expert witness. He founded the software engineering company Praxis in 1983 and was a coeditor of the NRC report referred to in this column. Thomas is a Fellow of the Royal Academy of Engineering and was appointed Commander of the Order of the British Empire for services to software engineering. Contact him at martyn@thomas-associates.co.uk.*

**Editor: John Harauz, Jonic Systems Engineering Inc., Willowdale, Ont., Canada; j.harauz@computer.org**