Alex Groce (agroce@gmail.com), Northern Arizona University

Nancy G. Leveson's *Engineering a Safer World: Systems Thinking Applied to Safety* is a serious book on a serious topic. It is, perhaps, the closest to a textbook of anything covered up to this point in the *Passages* columns. However, it is a textbook that most engineers of complex systems, and many "layfolk" interested in how to avoid catastrophic accidents as systems grow more complex should read. Sometimes, the best treatment of an important topic is not amusing or informal.

The subtitle of Leveson's book explains the key insight here. Through examination of numerous case studies, Leveson shows that traditional safety and root-cause analysis post-mortem methods for understanding accidents are fundamentally insufficient for the realities of the modern world. Cause-and-effect chains tend to understand accidents in terms of a series of dominoes toppling, which is sometimes adequate for grasping what amount to mechanical oversights, but is almost never a good approach when interactions between "correct" systems, or managerial/structural problems are involved in an accident or near-accident. Even when causes are simple, the "domino theory" tends to assign blame to one single cause, rather than the combination of multiple causes, and run into trouble if no single cause seems sufficiently blameworthy.

A second aspect of the insufficiency of approaches still typically used to analyze accidents is, for want of a better word, political, but connected to this limitation. Often, the purpose of an accident analysis seems more to blame some hapless Homer Simpson asleep at the wheel than to discover how such accidents can be avoided in the future. Even if operator error is foundational to a problem (which is seldom the whole story), a look at the system that allowed an incompetent operator to be hired and remain on the job would be in order, and spread the blame beyond the lowest worker on the totem pole.

For those with a background in software safety, some of the "greatest hits" of the field show up here. Part of my own thesis work concerned the TCAS collision avoidance system for civilian avionics, and reading this book prompted me to idle away a morning applying modern fuzzing techniques to the classic (and mysteriously, possibly, buggy) C code for that system: https://github.com/agroce/fuzz_tcas.

Other disasters referenced include the Bhopal disaster, a cornucopia of aircraft collisions and near misses, and the Challenger disaster. The reader of *Passages* may notice that most of these do not centrally, or even at all, involve software. That's true! The lessons here are broader than software, though addressing the new failure modes and complexities for analysis introduced by software is a core concern of the book. Software engineers can directly apply the lessons about incorporating safety into phases of a project beyond design and initial implementation, including especially operation and management (since few software systems are truly "self-operating" and almost all are maintained and changed by humans). Certainly awareness of assumptions is important to good software engineering, including critically how

assumptions *change* over time. Moreover, while the distinction between reliability and safety is a major theme of the early parts of the book, many of the lessons here are applicable to reliability as well, and the notion of safety used applies just as well to catastrophic failures that don't endanger human life; viewing major security breaches like aircraft collisions of nuclear meltdowns may be a good way to write software!

A particularly nice sub-theme is the design of user interfaces; the book's primary (perhaps only?) references to the infamous Therac-25, the most obvious "big software disaster" of all time, are in Chapter 9, which covers Safety-Guided Design. Leveson argues compellingly against simply using a human-factors checklist to do interface design, and emphasizes that a "usable" interface may be an unsafe one. The Therac-25 allowed operators, who were supposed to duplicate the entry of values at the radiation treatment site, to <PRESS ENTER> to confirm copying values from a computer. These values, due to a software error, killed people in some cases; had operators had to type them in again (which they had complained about), they might have detected the massive overdoses of radiation about to be applied. A similar point is that an interface that, during "lulls" in activity, is convenient for operators is likely to produce poor responses if suddenly an emergency arises. The example is too recent for Leveson's book, but consider the case of a hypothetical self-driving car that isn't good at handling sudden emergencies. A dozing human who has no need to interact with the system at all is unlikely to manage things well when summoned to action by a loud alarm. Instead, interface may have to keep operators on their toes, even at the cost of making humans perform actions computers could "do better."

For most software engineers, though, I think perhaps the most important idea in the book may be the repeated insistence on accident analysis as about understanding how to avoid an accident in the future, rather than simply placing blame. An analogy, without the labor-vs-management implications, can be drawn to the practice of placing the blame for a software failure on either a piece of code alone, or on the author(s) of that code. At minimum, though, a broader understanding of failure would include investigating why testing efforts didn't catch the problem. Was the oracle too weak? Was the code even executed (famously, almost any attempt to test Apple's "goto fail" bug at all, or even just examination of dead code warnings, would have detected the problem). If the problem is one of programmer incompetence, why were bad programmers hired and kept on the job? Regression testing, in my own field, at least avoids (or tries to avoid) precisely duplicating a past failure, the embodiment of Santayana's maxim. But too often the underlying QA practices that let a bug escape into the wild are never really analyzed, as soon as a suitable scapegoat is found.