Alex Groce (agroce@gmail.com), Oregon State University

Henry Petroski's *To Engineer is Human: the Role of Failure in Successful Design* is a curious thing: a replaceable classic. Instead of reading *To Engineer is Human*, you could read the more academic, more compressed, and more historically oriented *Design Paradigms: Case Histories of Error and Judgment in Engineering*. Or, if you're willing to disregard this column's rule that classics must be at least 10 years old, you could read *Success through Failure: the Paradox of Design*, which has the benefit (or possibly disadvantage) that it cares more about computers than either of the two older books, discussing PowerPoint in the first chapter. Last year Petroski wrote another book with a similar title, *To Forgive Design: Understanding Failure*, but I haven't read it. Chances are that it is more of the same.

What all these books have in common, other than an author and a strong overlap in words in the title, is a theme, expressed in the preface to *To Engineer is Human*: "the concept of failure ... is central to understanding engineering, for engineering design has as its first and foremost objective the obviation of failure." Petroski claims that failure is not only a central topic of engineering, but that the progress of engineering relies as much (or more) on understanding failures as on building upon success. Three (or perhaps four) of Petroski's seventeen books written to date are focused on this theme, and it appears as a persistent echo in his other books, including such popular tales of technological and design history as *The Pencil: A History of Design and Circumstance* and *The Book on the Bookshelf*.

According to *To Engineer is Human*, Petroski's obsession with failure started after the collapse of the Kansas City Hyatt Regency Hotel walkways in 1981, when a neighbor asked him how such a thing could happen, and rattled off the Tacoma Narrows collapse and other famous past (or expected future!) engineering disasters as other instances of engineers not knowing what they were doing. Petroski, as a professor of civil engineering at Duke, must have been as used to such harangues as most computer scientists are to our neighbors', relatives', and friends' litanies of the latest failings of their phone apps or desktop browsers. Rather than simply being annoyed, however, Petroski took on a lifelong quest to examine failure, understand it, and endorse the teaching of the history of failure as a core element of a real engineering education. He has pursued that quest with engaging, readable prose, and a deep knowledge of the history of engineering. Petroski's anecdotes are instructive and well-chosen (his books are, largely, composed of extended anecdotes), embellished with the references and connections available to a highly literate and literary mind.

Petroski's anecdotes in the core "failure books" often come from his own field, civil engineering. Computer science is, in his older books, not a real concern, though of course Petroski understands (and examines) the centrality of computer simulation to modern engineering efforts. Computer aided design mostly appears as a possible contribution to overconfidence in designs. On pages 199-200 of *To Engineer is Human* he discusses real world tests showing that the predictions of computer models of towers were considerably less faithful to the realized designs than engineers had assumed they would be. Many of his examples, particularly in *Design Paradigms*, however, greatly predate the computer. *To Engineer is Human* largely examines more recent disasters, but looks back to Galileo's *Dialogues Concerning Two New Sciences*. Why, then, should these books be classics to software engineers?

The answer lies in the title of the fifth chapter of *To Engineer is Human*: "Success is foreseeing failure." Petroski's books are not about civil engineering alone, or mere well-written histories of particular disasters in various engineering fields. Rather, they present a general claim about engineering and design. Namely, that the key effort of every design is to *avoid failure*. Petroski presents the design process as a task where the engineer must anticipate various possible modes of failure, and work to "answer" each "question" posed by these modes of failure. This view of engineering as a discipline is also compelling for software engineers. Testing and verification, of course, depend on a failure-centered view of software engineering. The field of formal methods is arguably largely devoted to the search for counterexamples, concrete proofs of design and implementation failure. Counterexamples are both more numerous and likely more famous than proofs of correctness -- consider the shock of the discovery that Needham-Schroeder Public-Key protocol was wrong, with a simple example of how it could fail. Security is typically pursued by enumerating and anticipating various threats. Static analysis tools work in large part by exploiting the relationship between certain modes of failure and certain properties of source code. Every programmer who has written even one *try/catch* construct has worked in the mode of anticipating failure and heading it off, driven by a knowledge of which failures are possible. Arguably, failure and improvement of design after failure is far more central to software engineering than any other discipline: only software engineers *expect* to produce failing products, and fix them based on the observation of field failure, as a matter of routine.

Petroski's historical approach (which has earned him the distinction of not only being the Aleksandar S. Vesic Professor of Civil Engineering at Duke University, but also a secondary appointment as a professor in Duke's top-15 ranked history department) admittedly provides few technical ideas for software engineers. I suspect Petroski's books provide few technical ideas for civil engineers, for that matter. What Petroski's books bring to the table is a look at the dangers of overconfidence and the value of knowing the past. While knowing how great bridges (including some designed by the greatest of bridge engineers) have fallen is unlikely to give software developers any immediate advantages in avoiding race conditions, knowing how bridge designers failed to take into account the *known failures of bridges in recent history* can motivate developers to *examine the recent failures of other programs and remember the failures of their own previous programs*.

Furthermore, Petroski's ideas support a concrete program that is also applicable to software engineering education. In the preface to *Design Paradigms*, he writes: "Historical case studies contain a wealth of wisdom about the nature of design and the engineering method, but they are largely absent from engineering curricula..." Software engineering curricula also tend to be focused solely on "the state of the art" and almost never include a significant element of historical case studies, or even detailed analysis of contemporary or very recent bugs that have resulted in software disasters. Too often, the only incorrect code students see in classes is their own. In many cases, software testing and debugging is not a required class for computer science students, or receives only a cursory examination as a one week unit of a class with another focus. Software design and architecture education often focuses on functionality and conceptual integrity, but avoids the difficulties of problems where handling failure is a major source of design complexity. Reading Petroski may not directly improve your ability to write correct software, but it may inspire you to reflect on failure and avoid hubris, which can only be to everyone's benefit. Moreover, reading Petroski is great fun and inspires a sense of wonder at the audacity of engineering, despite its endless failures. Today's failure is the foundation (if we let it be) for tomorrow's success.