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

Donald A. MacKenzie's *Mechanizing Proof: Computing, Risk, and Trust* is a highly readable and generally insightful history of, primarily, the effort to prove computer programs safe and correct. It won the Robert K. Merton (see the *Passages* column for October 2020) award from the Science, Knowledge, and Technology section of the American Sociological Association in 2003. It is a book of history and sociology, but is likely to also introduce some technical content that will be new to, yet of interest to, many software engineers.

MacKenzie's exploration starts with the issue of dependability: as soon as computer systems began to be seriously used in real-world applications, their reliability began to take on great importance. The book briefly but effectively sets the scene with the moment on October 5, 1960, when NORAD's early warning system seemingly detected a massive launch of Soviet missiles from Siberia (coincidentally on the same day that IBM's Thomas Watson, Jr. and other business leaders were touring the facility), and uses this chilling moment, caused by erroneous interpretation of the moon rising over Greenland, to lead into the first NATO Software Engineering conference.

While MacKenzie's focus is on attempts to prove programs correct (and critiques of these attempts), the book does a good job of placing that central dream of software engineering in the context of other efforts to make software work, including management techniques (through Harlan Mills), associated but not-proof-based methods (Dijkstra's crusade against goto and more generally the advent of structured programming), and of course testing. These alternatives to proof are treated fairly; if anything, the lack of focus on these methods means that their limitations are less on display than the limitations of proof.

At the heart of the book, however, is an examination not just of computer program correctness, but of the question of how we know even deductively "proven" truths. MacKenzie's sociological approach is a book-length, less polemic, elaboration of the points made in DeMillo, Lipton, and Perlis' (in)famous paper, "Social Processes and Proofs of Theorems and Programs." That paper, which Harry Lewis noted some readers had objected to his even including in *Ideas that Created the Future: Classic Papers of Computer Science*, openly attacked the plans of program verification advocates, and likely contributed to a "proof winter" in funding for the "strong" program correctness scientific program. MacKenzie does not simply endorse the view of "Social Processes" but places it in the wider story of the battle over the mathematical notion of proof, and how much it can ever be mechanized.

Chapter Four of the book therefore tells the full story, in considerable mathematical detail, of the four color conjecture, from the origins of the problem and early attempts to prove it, down to the controversial computer-aided proof that generally established the conjecture as true. Along the way, we encounter a first computer-aided proof of the conjecture, which in the end vanished when a second run of the program establishing the "proof" (in a roundabout way that had not been anticipated in the original run) failed to reach the same conclusion, perhaps due to a

hardware problem, perhaps due to being a run of a different, buggy version of the software. The more things change, the more they stay the same?

Other chapters of the book explore the early connections between automated theorem proving and AI research, and the role of the National Security Agency in program proofs. MacKenzie interviewed Dijkstra, Tony Hoare, Richard DeMillo, Robin Milner, Herbert Simon, Kenneth Appel, Bob Constable, Sherry Turkle, J Strother Moore, and other computer science and mathematics luminaries, and the broad range of opinions solicited yields a balanced and intelligent treatment of complex issues for which no simple solutions or unitary viewpoints are possible. I personally think some of the arguments of "Social Processes" for example, have been countered by type checkers, model checkers, and other "lightweight" methods, and (interestingly) other, deeper points are now being practically addressed by work in program mutation in part inspired by DeMillo himself. The arguments remain important, and by expanding its viewpoint to encompass both Dijkstra and DeMillo, Lipton, and Perlis (to take two arguably diametrically opposed camps within computer science), this book makes it clear how much the intersection of human trust and mechanized analysis matters.

A final note: one book constantly referred to throughout *Mechanizing Proofs* is the short masterpiece *Proof and Refutations*, by Imre Lakatos. I read this book, which had been recommended to me many times over the years, after finding these references extremely intriguing. I should have read it long before; any *Passages* readers who care about what it means for something to be proven, or how we refine the notion of a program's being "correct," and in particular who admire Popper, Polya, or Polanyi, should go out and read this extraordinary "Platonic dialogue in a classroom" immediately. If nothing else, you will never look at polyhedra in the same way again.