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

William Goldbloom Bloch's *The Unimaginable Mathematics of Borges' Library of Babel* is not actually the book I originally intended to write about for this issue. I had never heard of this book (published in 2008; finalist for the 2008 American Association of American Publishers PROSE award in Mathematics) until about two weeks before the deadline for this column, which is somewhat surprising, given the topic. I ran across the book while wandering in a library, naturally, and when I saw the title I was forced to stop and pay attention to it. My first reaction (beyond the initial interest in the intersection of math and Borges) was to be slightly annoyed at the title, thinking "unimaginable" to be sloppy (Borges imagined it, you lout!) and superficial. But, in fact, the precision with which the title was chosen is part of why I made this book this column's classic for software engineers. You see, Bloch says:

"Finally, about the title of the book: why the word 'unimaginable'? By way of an answer, we note that in his sixth Meditation, Descartes makes clear the distinction between simply naming a thing and visualizing it in a clear, precise way that allows for mental manipulations."

Bloch proceeds to quote Descartes' definition of the distinction, and in particular his idea that while we easily imagine a triangle (a figure of three sides) in a concrete mental image, our equally formally well-made notion of a chiliagon (a figure of a thousand sides) is not visible to "the eyes of [our] mind." This point is the bridge to software engineering: every sufficiently complex computer program is a kind of library of Babel, with a state space and set of behaviors that is, in this formal and literal sense, "unimaginable." The last sentence of Bloch's introduction therefore serves as a motto for those of us who would train others to be good software engineers: "our task as your guide is to trigger the processes by which you build intuition and insight into the Unimaginable."

Before proceeding, let's summarize the story.

```
import itertools

# The 25-character alphabet
alphabet = list("ABCDEFGHIJKLMNOPQRSTUV .,")  # 22 letters + space + comma +
period

# Book format constants
PAGES = 410
LINES_PER_PAGE = 40
CHARS_PER_LINE = 80
CHARS_PER_BOOK = PAGES * LINES_PER_PAGE * CHARS_PER_LINE  # 1,312,000

def generate_books():
    book_number = 1
```

```
    # itertools.product creates the Cartesian product (like nested loops)
    for book_chars in itertools.product(alphabet, repeat=CHARS_PER_BOOK):
        char_index = 0
        for page in range(PAGES):
            for line in range(LINES_PER_PAGE):
                line_text =
''.join(book_chars[char_index:char_index+CHARS_PER_LINE])
                print(line_text)
                char_index += CHARS_PER_LINE
        print(f"END OF BOOK ({book_number})")
        book_number += 1

# Start generating books (infinite loop in practice)
generate_books()
```

Well, you may, think, that's an odd summary. Borges' story concerns a library, composed of hexagons with bookshelves; each book on the shelf has 410 pages of 40 lines of 80 characters each, using an alphabet of 22 characters, plus comma, space, and period. The library is total.

It is that last point that compels this book. This library, which contains 25^1,312,000 books, must, even assuming very modest dimensions (far too small a print for my weary aging eyes) for each book, be vastly, profoundly, larger than the most absurd estimate of the size of our universe, assuming our universe to have some finite size, rather than being unbounded.

Bloch's book, written by a literarily-inclined mathematician, makes the connection between this construct and computer science explicit; in one chapter, he proposes that a librarian is a Turing machine, in fact. And in so doing, though he does not make this connection explicit, he hints at the transformation embedded in the program shown above (or the similar programs Bloch mentions at various points, for generating the library's contents): the transformation of unimaginable size into unimaginable time.

The time is explicit above; the program will, in fact, not finish running; proton decay or the heat death of the universe will get it, even if somehow your MacBook survives a few billion years. But the time can be made even more implicit, in that a simpler program can produce a single randomly chosen book from the library (it's easy to reconstruct once you see the central piece of code will look something like:

line_text = ''.join(random.choices(alphabet, k=CHARS_PER_LINE))

This version of the concept reminds me of R. A. Lafferty's story "Been a Long, Long Time," where modestly sinful angel causes some problems at the beginning of time, and is sentenced to test out the theory that six immortal monkeys could, by happenstance, produce the works of Shakespeare, given typewriters and time. The archangel Michael helpfully provides Boshel with

a clock, a stone cube one parsec on each edge. As Michael notes, it's a very nice clock. "You don't have to wind it, you don't have to do a thing to it" since "A small bird will come every millennium and sharpen its beak on the stone. You can tell the passing of time by the diminishing of the stone. It's a good clock and it has only one moving part, the bird." To help think about periods of time that are actually large (unlike the life thus far of the universe, or plausible future lifetimes of the universe, both very small times indeed; Boshel sits through many universal lifetimes, and then some), I strongly recommend hunting down this wonderful story. Lafferty's collection *Ringing Changes* is probably the easiest way to find it physically; the only electronic book versions of the story I am aware of are in "bootleg" Lafferty editions.

The probabilities poor Boshel tangles with resemble those of producing any single, exact, behavior of a computer program taking significant inputs in the process of random testing or fuzzing. What distinguishes those of us hunting for bugs from the hapless Borgesian librarian or Lafferty angel seeking a particular book among the Books is that there are usually an unimaginably large set of executions of a program that will trigger a particular bug, so we are happy to, metaphorically, find any of the myriad myriad myriad books that contain, e.g., the phrase "SO LONG AND THANKS FOR ALL THE FISH." Note that the improbability of finding such a book physically still means that in practice no librarian wandering the hexagons will ever stumble across one, but we have computers and coverage-driven feedback and SMT solvers so we can cheat a bit. On my computer, using nothing but the sheer speed of modern computation, finding a book with "Borges" in it typically takes less than

Bloch's book covers other, less computer-science-relevant mathematical penumbra of the Borges story equally well, and the possible topologies of the library in particular are delightful to fail to imagine. The exploration of the sheer size of combinatorial reality, however, is the first and core joy of this work for a software engineer, and serves to make it clear how extraordinary it is that we have tools (software tools and conceptual tools) that allow us to think about, analyze, and effectively test computer programs' whose actual space of possible states is, in some cases, much larger than the library of Babel. The next time you're running a fuzzer or a static analysis tool, pity the poor librarians at work hunting for "good reads" in your code.