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

Hugh Kenner's *The Mechanic Muse* is a peculiar book. On the one hand, it is (maybe) most usefully read by people who are well versed in modernist literature, the oddballs who cannot hear the word "stately" without a vision of plump Buck Mulligan from *Ulysses* appearing before them. On the other hand, it may appeal most to people whose admirations and notions are more typical of the thoughtful engineer, scientist, or software developer than the average student of 20th century literature. Let's start over.

Software engineering is a peculiar business. On the one hand, it is partly the enterprise of placing "words," in a certain order, with certain semantic properties resulting from their syntactic (or even typographical) arrangement. On the other hand, it is partly a matter of the construction of *engines or devices* (as the Victorians might say) as in traditional mechanical engineering. Software engineers are neither writers nor machine-makers, but we spend most of our time writing and most of our time building complex "machines" that *do things*.

The Mechanic Muse is not the most obvious book for a busy software engineer to pick up. I understand if you don't run out and buy it (for one thing, it's not in print). Because it is a peculiar choice, and because I couldn't figure out where to start in selling this masterpiece to you, the remainder of this column is in Q&A format, an interrogation of the columnist by a skeptical but interested software developer friend who has stopped by my office and seen the book lying on my desk, and read the column's beginning that you just finished reading.

Q: What's this book about, anyway? You haven't said yet. Like most of Kenner's books, it's about a whole bunch of things. That's why I like it. But the short answer is that if you had to shelve it in a library, it would belong in the "literary criticism" section, and the topic is (mostly) four major modernist writers: T. S. Eliot, Ezra Pound, James Joyce, and Samuel Beckett. More specifically, it is about how the *machinery and technology* of the 19th and 20th century created the assumptions and structures of their work. One big idea here is that the methods these writers used were also technologies or inventions, just like the trams, linotype machines, typewriters, and wristwatches that inspired them. The sections of the book are titled: 1) "In Memoriam Etaoin Shrdlu," 2) "Eliot Observing," 3) "Pound Typing," 4) "Joyce Scrivening," 5) "Beckett Thinking," 6) "Epilogue," and 7) "Appendix: Science, Axel, and Punning."

Q: Huh. Why would I read that? I've read a little of Eliot's poetry, and I guess I thought *Waiting for Godot* was cool, but I want no part of James Joyce and I'm not even sure who Ezra Pound is. Isn't he in some Bob Dylan song? Yes, there's a Bob Dylan song. Anyway, the reason you should read this is that it's a brilliant, thought-provoking, fun, book. It's short, too, not even 140 pages, counting the appendix. There are reasons *you* particularly might want to read it *as a software engineer*, though.

Q: Such as? Look up top of this here *Passages* column. Kenner's book is about the point

where literature -- writing words down in a certain order to some purpose, if you want to look at it pretty broadly -- touches technology, the workings of complicated machines. What else is software engineering but just that? Programs are machines that do things, and they are written communication for other people to read. Moreover, just as in literature, you can say the same thing in many different ways, and it matters how it is said. Think about Perl and Python not as competing scripting languages, but as different literary modes for expressing similar stories. Kenner's book gets you *thinking* about this kind of thing. In fact, he talks about two different kinds of books.

Q: What two different kinds of books? Kenner implies their existence in talking of two kinds of scholars. One kind "is someone who knows a non-scholar is wrong, and need not say how he knows it." The other kind of scholar (the kind Kenner was) has a different concern: "to help you know what he knows." Ezra Pound talked at length about poems (and machines, and coins) as concentrations of energy, and Kenner says that books by the good kind of scholar are also concentrations of energy, to save someone else the time they spent learning what is in such a book. Books are like machines, like computer programs, then. And one thing this book does that should please any software engineer is that it offers a grand defense for time-saving, whether by the typewriter, or the tram, or the computer program. A book like this teaches all kinds of odd and connected and important things, without being easy to put in any one place on the shelf. It gets you thinking.

Q: Thinking things like that idea about Perl vs. Python? Right. Dickens and James Joyce to a large extent can tell similar stories (they don't, exactly, but they could). But the way these writers "tell a story" is very different. That's something Kenner talks about. Dickens is basically using typeset words only as a convenient way to permanently record the voice of a storyteller. His books were meant to be read aloud, ideally by himself. Joyce is *writing for the printed page*, the typeset page. If you see an "error" like "did she wrote it herself" in *Ulysses* it is likely to be intentional, meaningful, and possibly even important. In a copy of Dickens, it is just a bug, if it is in the narrator's voice, and not a quotation from a semi-literate character. Dickens is conversational, with lots of help for the lost reader. James Joyce requires practice and work to read, but this is not without its advantages, once you are up to speed, though you must pay attention to each individual letter, not just the vague shape of a sentence. Think about the difference between a COBOL program and a C program (or a Haskell program, if you like). That's not totally unlike the difference between Dickens and Joyce. Now, the costs and benefits here are different than in programming, which is why many novels are still written in "Dickens" but not many programs are written in COBOL. But there are similarities worth thinking about.

Q: I'm not convinced. What else have you got? Look, it's a suitable book for computer people, trust me. There's a Pascal program on page 94 of my copy, look at it. That's a Pascal translation of a paragraph from a novel (of sorts) by Samuel Beckett. Good enough? How about this, which is Kenner's quotation of a character in Beckett, and also his explanation for why programming is difficult: "Ah the creatures, the creatures, everything has

to be explained to them" (in a paragraph that ends with the words "helped debug a system").

Q: Who was this Kenner guy anyway? Some kind of English professor, right? Why is he writing Pascal programs? He was some kind of English professor, alright. He wrote the definitive book on the modernist movement, *The Pound Era*. He was also a columnist for *Byte* magazine for many years, and he wrote the user's manual for the Zenith Z-100 computer. He wrote a book about geodesic math, and one on Chuck Jones, the guy behind most of your favorite Road Runner and Bugs Bunny cartoons. His mentor in grad school was Marshall McLuhan, of *Understanding Media* and *The Global Village* fame, that odd Canadian prophet of the Internet. Kenner was into a lot of things, and computer programs are *very* interesting things to the right kind of English professor. Modernist literature may be very interesting to a certain kind of software engineer. I want to add that there's nothing pretentious, condescending, or pedantically dull in Kenner's books. Reading Kenner is like reading Fred Brooks: it's just a conversation with a friendly man who knows a lot you'd also like to know.

Q: You really think I'd like it? You might. You might not. I left out most of the good bits in this one, which starts off with an introductory chapter that takes us from ETAOIN SHRDLU to Silicon Valley and MIT. George Boole (of Boolean algebra and our favorite type) has an index entry. The four-color theorem proof shows up. My favorite sentence is this one: "Sextus Empiricus, who flourished circa A.D. 190 (it gets put that way, though his tone seems not that of a flourisher) was a physician whose three volumes of *Pyrrhonean Sketches* tended to reduce all knowledge to ashes." Every page something new and unexpected is right there in front of you, and I guarantee you can re-read the whole thing, with profit, once every two years if it turns out to suit you at all. I don't promise reading *The Mechanic Muse* will make you a better software engineer, but I think it is likely to give you some new ways of thinking about the problem of writing something down for other people to read (and how hard that is, and how strange that we should ever think it could be easy) *and at the same time* building a complicated machine that goes out and *does something*.

Q: I'm sold. Do I have to read James Joyce, though? Couldn't do you any harm, buddy.