language." "This puts severe restrictions on the types of debugging procedures that may be used, and, in fact, restricts them to debugging at the compiler language level." Such judgment was harsh and shortsighted. In 1963, several systems had symbolic debuggers, software that would connect user programs to the operations of the machines. Within a decade, symbolic debuggers would be common on most computers.

During the 1980s, the debugging literature expanded for a second time with the introduction of more sophisticated programming ideas such as concurrency, parallelism, distributed programming, and optimizing compilers. For programs that used these ideas, the old symbolic debuggers no longer gave a clear picture of machine operation. Such ideas complicate "the mapping between the source code and the object code due to code duplication, elimination, and reordering," observed one research team. They also make "reporting values of source variables either inconsistent with what the user expects or simply impossible."

However, the problem of connecting a program to the machine's underlying state is only a small part of debugging. The bigger task, that of hypothesizing the machine's state and reasoning from the actual state back to the program's logic, defies a general engineering solution. "Debugging," wrote one author in 2001, remains "as labor-intensive and painful as it was five decades ago."

Some researchers have developed algorithms to handle some of the debugging work or to automate the task of checking all possible paths through the code, but the basic work remains unchanged. It is no easier to find the bugs in a program than it is to discern the activities inside a home by standing on the lawn and observing the shadows inside the windows.

At Bagley Avenue Systems, the debugging took four days. The hardware was the easiest part to fix. The software was considerably harder. It controlled two processors, one that handled the incoming data and another that prepared the output.

At each step of the code, Kai demanded to know how the machine was to transform the flows of information and asked Les to demonstrate that the processors were behaving correctly. Les responded to each request, but he was no more patient with this process than he had been during his initial encounter with Kai.

As they came to the end of the debugging, they were no closer to working as a team than they had been on the first day. Kai continued to make his requests, often punctuating his questions with comments that irritated Les. Les did all that was asked of him, but he still resented the idea that he was working with someone who couldn't understand the details of his design.

By the end of the last day, the rolling stars had vanished from the oscilloscope. In their place were a gentle wave and a few dots that would appear at random moments and quickly vanish. Les was not pleased with the picture and turned on his partner. "You've added a new error to the code," he claimed in frustration. "It's supposed to be straight."

"Prove it," Kai responded.

Les pushed a pile of cables to one corner of the table, found a piece of paper, and started to write the equations that described the system's operation. When he reached the end of the page, he started to search for a second sheet, but Kai stopped him. Pointing to one line, Kai said, "This statement is only an approximation."

"But it's good enough," blurted Les.

"Perhaps," said Kai, "but if you add a second term, you'll see that there's a wave in output."

Kai paused to let Les fully comprehend the issue. Then he said, "If you add a third term ..."

"What ..." Les interrupted.

"If you add a third term, you'll see that it is sensitive to small changes." Kai hesitated and then corrected himself, "Sensitive to dirty electricity." ■

David Alan Grier *is the editor in chief, IEEE Annals of the History of Computing, and the author of When Computers Were Human (Princeton University Press, 2005). Grier is an associate professor in the Center for International Science and Technology Policy at the George Washington University. Contact him at grier@gwu.edu.*

Note on the January column:

Russell Kirsch of Portland, Ore., has written me to say that the National Bureau of Standards did not terminate most computer research in 1954, as the column implied, but continued to be a major source of innovation for at least the next two decades.

Mr. Kirsch is indeed correct. The unit that was terminated in 1954 was the Machine Development Laboratory in the Mathematics Division. This group oversaw the design and construction of four major machines for the US government between 1948 and 1954, including the SEAC and the Univac I. It was less a true research laboratory and more a contracting unit. "There is no way to learn how to build computers except by building them," noted one lab report. By 1954, other parts of the government knew how to oversee the specification and design of machines, hence there was no need for the Bureau of Standards to provide that service.

Mr. Kirsch also noted that he worked with Ida Rhodes and that she was doing programming well into the 1960s. His comments reminded me of an interview that I had long forgotten in which a coworker characterized Rhodes as a "genius programmer."

I am most grateful for Mr. Kirsch's comments and welcome the comments of other readers.

—David Alan Grier