Giving a sigh of obvious contempt, Les took Kai over to the oscilloscope and showed him the picture of little green stars rolling through the luminiferous ether.

"What do you think you can do with it?" Les asked the consultant.

"Depends," Kai replied, "on what it's supposed to look like."

"A straight line," Les responded.

The two stared at the screen in silence. Les took comfort in the idea that Kai's tenure would probably be short.

"What do you think is wrong?" Kai finally asked

"Everything is perfect," Les stated confidently. "I've checked every element of the circuit and looked at every line of code." Using a term that normally describes a power source that occasionally has random flucuations, he said, "I believe the problem is dirty electricity."

"Hmmmp," replied Kai. "So you think the electricity passed through a porn video on its way into the house?"

This remark irritated Les, but before he could respond, Kai asked a second question. "Are you sure that the input to the device is correct?"

Feeling that he was being mocked, Les decided to put Kai in his place. He uttered a well-known expletive, disconnected a cable, flipped a switch, and turned back to the oscilloscope. A green sine wave appeared on the screen.

"Is that the right signal?" Kai asked.

"No," said Les.

"Good," said Kai. "Let's begin here. It's time for us to remove the bugs."

AN OLD CONCEPT

The concept of the "bug," a small error in design or implementation, can be traced back to the earliest days of the computing science field. In February 1944, a staff member in Howard Aiken's Harvard laboratory wrote that they were running test problems on the electromechanical Mark I to "find bugs." In a story that Grace Hopper recounted, Aiken's staff actually found a moth in their second machine, the Mark II, and taped it in their log book with the notation, "first actual case of a bug being found."

In fact, the term is much older than the computer era. Thomas Edison used the term as early as 1878 to describe problems with electrical systems. "Bugs," he wrote to a friend, "show themselves, and months of anxious watching, study, and labor are requisite before commercial success—or failure—is certainly reached."

The debugging anomaly

Debugging, the process of removing conceptual errors form programs, has been one of the crucial skills of computer science. Only in those brief days of naiveté that followed the introduction of high-level programming languages did anyone profess that a computer system could be built without hours of debugging, yet debugging has held an anomalous role in the field.

"Program testing and debugging occupies more than 50 percent of a professional programmer's time," complained an early author in the field. He noted that the process of debugging a program was "time-consuming, difficult, poorly planned, and often slighted." At the same time, he observed that most computer scientists could offer little guidance on the subject. "Students and teachers are encouraged to minimize errors by having good work habits, being methodical and neat, and checking for clerical details. This is certainly good advice, but it doesn't really help when it comes to fixing a problem."

The topic of debugging has never commanded enough attention to create its own comprehensive theory, its own journal, or even a regular conference devoted to the subject. Indeed, the theoretical literature has largely ignored debugging. The journals devoted to computer theory have published only a couple dozen articles on the subject in the years since the ENIAC began operation in 1948.

Debugging receives better treatment in the applied computing literature, but even in these journals, the coverage is incomplete. "Debugging is said to be the least established area in software development," observed the authors of a 1991 article. "Industrial developers have no clear ideas about general debugging tools, yet they debug programs anyway."

An empirical task

One aspect of debugging that separates it from other aspects of computer science is the fact that it is an empirical task rather than a synthetic activity. Good debuggers must reason from effects back to causes, a process that is fraught with problems. They must build a mental model of how the machine should behave under the control of the program and hypothesize the machine's state at points during the execution of the code. From observing the machine's actual state and comparing it to their hypothesis, debuggers must assess their model and determine how they should adjust their ideas and then adjust the code itself.

Much of the debugging literature has presented ideas for software tools that would help with one part of the process, the part that connects the code to the machine's state. Articles on such tools began to appear in the 1960s when high-level computer languages began to separate the programmers from the machines. The authors of a 1963 article stated, "It appears that many programmers of large-scale computers today do not understand and often do not even wish to understand symbolic machine

IEEE Annals of the History of Computing

The *IEEE Annals of the History of Computing,* the IEEE Computer Society's historical magazine, has published several articles on bugs and debugging. The current issue of *Annals,* available in the Computer Society's Digital Library (www.computer.org/annals), leads with an article on computer crime.