Name: _____ Date: _____

Debugging Since 1843

Reading level 13 · 1449 words

In 1843 Ada Lovelace published a set of notes on a calculating machine that would never be built. The last of them, known as Note G, set out a table of steps by which Charles Babbage's Analytical Engine could compute a series of values called Bernoulli numbers. It is often described as the first published computer program. Modern programmers have since tested that table, and they found something that anyone learning to code should find comforting. It contains a bug. In one division, two variables are swapped, so if the engine had ever run the steps as printed, it would have produced the wrong answer. The first program, if that is what it was, did not work on the first try. Hardly any program ever has.

Lovelace was also, on Babbage's own account, a capable debugger. He later wrote that when he worked out the Bernoulli math for her, she sent it back for correction, having found a grave mistake in it. Her credit is genuinely debated. The historian Allan Bromley pointed to several dozen simpler sample programs that Babbage had prepared between 1837 and 1840, which were never published. Biographers still argue over how original her contribution was. What is not in doubt is the idea she put into words. She suggested the engine might act on things other than numbers, and might even compose elaborate music. That leap, from a machine that calculates to a machine that follows rules about symbols of any kind, describes what every computer now does.

Babbage did not invent the notion of a machine that obeys a stored sequence of instructions. He borrowed it from weaving. In 1804 Joseph Marie Jacquard patented a loom controlled by a chain of punched cards laced together, with the design of the fabric stored on the cards. Change the cards and the same loom wove a different design. One famous portrait of Jacquard, woven in silk on such a loom, needed 24,000 cards; Babbage owned a copy, and Jacquard's punched cards inspired the ones he planned for his engine. A deck of cards is a program in the most literal sense. It is a sequence of steps written outside the machine and fed into it, and a single misplaced hole will be woven into the cloth, faithfully, every time.

The word for such a flaw is older than computing. Engineers were calling small defects bugs by at least the 1870s. In a letter in 1878, Thomas Edison described how every invention of his began with a burst of intuition, followed by difficulties, at which point bugs, as such little faults were called, showed themselves. Months of watching and labor would follow before he knew whether the thing would work. That sentence describes most programming projects rather well. The idea arrives in a moment, and then the real work begins, which is the slow business of discovering every way the idea fails. Edison was not writing code, but he was describing a workflow that programmers would recognize a century and a half later.

The most famous bug was an actual insect. On 9 September 1947, operators testing the Mark II computer at Harvard University found a moth trapped between the contacts of relay number 70. They taped it into the machine's log with the note "First actual case of bug being found." The joke depends on knowing that bug already meant a fault, and the operators were probably making a pun. Grace Hopper, who worked on the Mark II, later made the story famous. Sources disagree about one detail. The US Navy's history page says the operators introduced the phrase debugging a computer program, while other accounts stress that the engineering use of bug was long established. The Navy's claim is the shakier one: by 1945, engineers were already writing about debugging airplane engines.

On the earliest electronic computers, debugging was physical work. ENIAC, completed in 1945, weighed more than thirty tons and stretched about a hundred feet. It had no programs stored in memory. Instead, it was set up with plugboard wiring and banks of switches, and once a problem had been planned on paper, setting the switches and cables could take days. Most of that programming was done by six women: Kay McNulty, Betty Jennings, Betty Snyder, Marlyn Wescoff, Fran Bilas and Ruth Lichterman. They learned the machine's inner workings so thoroughly that they could often trace a bug to a single failed vacuum tube. To fix a program, you sometimes had to find one faulty part among thousands.

Grace Hopper's great contribution was to move programming away from the machine and toward the person. She argued that programs should be written in a language based on English and then translated into machine code by a separate program, which came to be called a compiler. She was told, quickly, that this was impossible because computers did not understand English. By 1952 she had a working program she called a compiler, though today it would count as only one piece of a compiler, a linker. She later recalled that nobody believed it and nobody would touch it. In 1959 she helped create COBOL, a language built on English words. The underlying fact has not changed. A processor runs only its own machine code, made of binary digits. Every language a person types is a translation layer, built so that people can read what they wrote.

Readable code matters because a mistake you cannot read is a mistake you cannot fix. Python shows how far this idea can be pushed. Guido van Rossum began it during the Christmas holidays of 1989 and posted it publicly in February 1991. He took a key feature from an earlier language called ABC: the indentation at the start of each line is not decoration but meaning, because it tells the computer which lines belong together. The visible shape of the code therefore matches its logic, so a line that is out of place usually looks out of place. Van Rossum also meant the language to be fun. He named it after the comedy series Monty Python's Flying Circus, and some Python examples still say spam and eggs where other languages say foo and bar.

Scratch, released by a team at the MIT Media Lab in 2007, goes further still. Its programs are built from blocks that snap together, so a whole class of bugs, the missing bracket and the

misspelled command, cannot happen at all. What remains are the bugs that matter most for learning, the mistakes in logic, such as a check placed outside the loop that needed it. Scratch also built a habit into its design. Its name comes from DJs scratching records to mix sounds, and its users remix one another's projects, opening them to see how they work. Opening someone else's work to see how it runs is an old habit among programmers, and Scratch makes it the first thing a beginner does. And the habits formed in Scratch, such as loops and conditions, carry over to languages like Python.

Programmers have even developed rituals for finding bugs. The first step is usually to make the problem happen again, on purpose, so that it can be studied rather than merely feared. A second is explaining the code aloud. A well-known book for programmers, The Pragmatic Programmer, gave one of these habits its name: rubber duck debugging. The name refers to a research assistant who carried a rubber duck and explained his problems to it, step by step. The act of explaining often makes the problem leap out, and talking to an object seems to work better than simply thinking aloud alone. None of this is trivial. A 2002 study for the US government estimated that software bugs cost the American economy about $59 billion a year. Finding mistakes is not a side task in programming. It is a large share of the job.

So what does this history say about how to learn to code? The languages keep changing. Punched cards, plugboards, COBOL, Python and Scratch have each had their turn, and newer tools will replace the last of these. The skill that survives every change is the one Lovelace used on Babbage's math. You read your own instructions the way a machine will read them, with no sympathy for what you meant, and you find the place where the two diverge. Even Note G, the work of both of them, kept a swapped pair of variables, and the error waited until the age of modern computers to be found. That is the honest starting point for any beginner. Write something small, run it, and expect a bug. Then go looking for it, preferably with a duck.

Questions

1. What did modern programmers find when they tested Note G?

A. Two variables swapped in a division
B. It was written by Babbage alone
C. It had no steps at all

2. Why does the essay say the 1947 moth note was a pun?

A. Moths were rare at Harvard
B. People already used bug to mean a fault
C. The computer was named Moth

3. How did Jacquard's loom connect to computing?

A. It used electricity

B. It was controlled by a chain of punched cards, like a program

C. It could do arithmetic

4. In the last screen, what does "diverge" mean?

A. Join together

B. Speed up

C. Go in different directions

5. Why does the essay argue that readable languages matter?

A. They run faster

B. A mistake you cannot read is a mistake you cannot fix

C. Computers prefer English

6. According to the essay, which skill lasts through every change in languages?

A. Memorizing one language

B. Typing quickly

C. Reading your own instructions as a machine would and finding mistakes

Words to know

relay: electric switch
compiler: code translator
indentation: line-start spaces
debugger: mistake finder
diverge: go different ways
pun: word joke

Answer key: Debugging Since 1843

1. A. Two variables swapped in a division
2. B. People already used bug to mean a fault
3. B. It was controlled by a chain of punched cards, like a program
4. C. Go in different directions
5. B. A mistake you cannot read is a mistake you cannot fix
6. C. Reading your own instructions as a machine would and finding mistakes