1969

Debunking the Computer Mystique

Reed Dickerson

Indiana University School of Law

Follow this and additional works at: http://www.repository.law.indiana.edu/facpub

Part of the Computer Law Commons, Legal Writing and Research Commons, and the Library and Information Science Commons

Recommended Citation

Dickerson, Reed, "Debunking the Computer Mystique" (1969). Articles by Maurer Faculty. Paper 1520.
http://www.repository.law.indiana.edu/facpub/1520

This Article is brought to you for free and open access by the Faculty Scholarship at Digital Repository @ Maurer Law. It has been accepted for inclusion in Articles by Maurer Faculty by an authorized administrator of Digital Repository @ Maurer Law. For more information, please contact wattn@indiana.edu.
DEBUNKING THE COMPUTER MYSTIQUE

Reed Dickerson*

Because a dull subject needs a provocative title, the title and description of my particular contribution, as shown in the printed program, were assigned to the author of the ecstatic menus at the Marriott Motels. On this occasion he outdid himself. Just reading the description of it gives me goosepimples.¹

Let me begin with a paradox: I was selected to debunk the computer mystique, not because of my knowledgeability respecting computers, but because of my lack of it. You see, the problem has not been telling the lawyer too little but telling him more than he can assimilate. My ignorance of the computer is so profound that I was selected as superbly qualified for this assignment because I had the closest affinity to the uninformed lawyer. Unfortunately, I may wind up adding to the mystery instead of reducing it.

The average lawyer's uneasiness when confronting a computer results largely from an unusual condition. Whereas his needs usually precede their solution, we have here important legal needs emerging at a time when mature and highly developed contraptions already exist. This explains David Henkle's observation that in the computer the lawyer is being confronted with the immediate maturity of technological solutions, Arnold Dumey's remark that we have here "a solution in search of a problem," and Lee Loevinger's comment that "invention is becoming the mother of necessity."

Our problem today is to match the most appropriate approaches from a wide array of technological solutions with the specific needs of lawyers that are now rapidly emerging. The impressiveness of this complex of potential solutions is probably what most frightens or baffles lawyers.

One reassurance I have is that, far from being an intelligent demon in a mysterious container, the computer is, like the dinosaur, very big and very stupid. Many of those who know computers best have referred to them as "idiots" or "imbeciles." Left alone, a computer is a helpless cripple. Even with guidance, they are, when compared to human beings, woefully inefficient. Routine operations that humans perform directly and easily computers perform in circuitous ways. I will give examples later.

* Professor of Law, Indiana University; Chairman, Standing Committee on Law and Technology (1968-1969).

1. "Debunking the Computer Mystique. Explanation of the working of the computer in layman's terms for those who do not have a real understanding of what a computer is, how it works, and what its limitations are. This presentation will strip away the mystique that has enshrouded computers and computer operations, and expose the computer as a relatively dumb machine capable of tirelessly performing only the simplest mathematical chores and only after suitable instruction."
A more serious limitation in computers is that they have no insight or intuition. They can only respond to specific instructions. Basically, they can only manipulate symbols and the syntactical relationships among symbols. For this reason, they can make no semantic judgments and thus cannot detect or deal with vagueness or ambiguity. They cannot even detect misspellings, except as specifically instructed. To a computer, a misspelled word is a different word.

Although I have said some uncomplimentary things about digital computers, they have several compensating assets. One is blinding speed. Another is invulnerability to the fatigue factors that plague human beings. Third, because no computer suffers from arterio-sclerosis, there is nothing to impair its memory. It never forgets. These three important capabilities contribute to a total capability for dealing superbly with many types of routine, particularly complex routine.

Let me now make a few, possibly helpful, over-simplifications about computer operations and their supporting hardware. In general, there are three basic phases in computer operations. The first is getting the initial information, called “input,” into the computer. The second phase consists of the operations that the computer performs on that information. The third phase is getting the final result, called “output,” out of the computer. Corresponding to these three kinds of operations are three kinds of hardware.

There are a number of input devices, which I will mention but not describe in detail. These are the devices that take human information and put it in a mechanical or an electronic form that can be taken into the computer’s memory, where it is available for internal processing. You are familiar, of course, with punched cards. Sometimes the input device is, instead, magnetic tape or paper tape. It might even be a special typewriter or an optical character recognition device. The details are unimportant. The main thing is that there are devices that pick up the information in human form and pop it into the computer.

In the internal operations phase, the first element is the computer memory, which receives and stores the data that it is going to work on later. It is like a storage bin or room. It consists mainly of doughnut-shaped cores or thin film elements supplemented by auxiliary storage devices such as magnetic tape, discs, tape strips, or drums, some of which (magnetic tape and some disc devices) can be removed. Each has its advantages and disadvantages. Core memory is very fast, but limited in capacity and very expensive. Magnetic tape is relatively slow, but large in capacity and relatively inexpensive. The processing of data under the control of a program goes on in the core memory, not in the auxiliary storage devices.

The memory holds several things. Besides the data input, it holds the operations programming, which is a set of instructions that tell the computer generally how to go about its job. (These are left in the machine.) Finally, it holds the special instructions that the user puts in when he is solving a particular problem. Because these are temporary, they are included as part of the search question asked. In short, the computer’s memory contains the substantive information that is permanently stored, the permanent general instructions, and the temporary specific instructions.

Besides the memory, there is a unit associated with memory called the “control unit.” This takes the material to be processed out of memory to a location where it can be worked on. This also controls the order of operations. There is also an associated processing unit called the “arithmetic” (or “logical”)
unit, which works on the material brought over from the computer memory for processing. Together, they look something like this.

**BASIC COMPUTER COMPONENTS**

![Diagram of basic computer components]

Through the control and arithmetic units, a computer can perform a number of fairly elementary functions like counting. It can also compare numbers. For example, it can tell whether one number is the same as, or greater or less than, another number. But it does this very inefficiently. If you and I were comparing paragraphs, we could see, for example, that two sentences were alike or different. A computer can't do that. It can't even compare whole words. It can compare words only letter by letter and numbers only digit by digit. In view of these limitations, there is little reason to feel overly modest when you are in the presence of a computer.

Another basic operation that a computer can perform is to "branch." It can come to a fork in the road and decide to go this way or that way. It does this through its ability to throw a switch.

Building on these basic operations, the computer has developed a capacity to sort and arrange. This is a tremendous capability. It is important, for example, in building indexes, because sorting and arrangement, particularly where many search terms are involved, is a heavy mechanical burden.

Finally, the computer can in effect add, subtract, multiply, divide, and otherwise calculate in various ways. But it can't do these things directly. It can't even add directly. For instance, a computer can't even add 2 and 2. But it can count. It takes two units here and two there and then it counts up to four. Also, a computer can't multiply directly; it multiplies by counting in cycles. Thus, if we give the computer the problem of determining the product of 3 and 7, it handles it by counting up to 3 seven times. Subtraction and division, which for us are elementary operations, are performed by reversing these elementary counting operations.

After a computer has worked on its material, the final results are disgorged through an output device in the form of visually readable print-outs. Output devices include slow-speed typewriters, high-speed printers (1000 lines of 132 characters a minute), and microfilm printers (30,000 characters a second). Although these devices are slower than the other operations of the computer, the printers are still relatively fast.
The important thing in dealing with a computer is to put into it not only the right information but the right instructions. These instructions are necessarily detailed, and they must be correct. The computer's performance can be no better than its instructions.

In programming, we look for a procedure to solve a problem, a procedure that is framed in terms of the computer's specific capabilities. Once such a procedure is developed, it must be put in the kind of special language, such as Fortran or Cobol, that a computer can understand.

Essential to programming is the use of a flow chart, block diagram, or logic diagram. Here, the problem must be analyzed and broken down into its constituent steps, and these steps must then be arranged in logical order. These steps are tiny ones, very specific and very precise.

To get the general idea across, let me talk in very simple terms. Taking an elementary non-legal problem, suppose we want to know the answer to the question, what is 50 times the sum of 10 and 20?

The computer works in a round-about way. It doesn't solve the whole problem directly. Instead, it solves a small piece of it and then repeats the operation a number of times. Here, the computer can solve our problem only by adding 10 and 20 (by counting up to 30) and then repeating the operation 50 times.

Because it's going to repeat the cycle of adding 10 and 20, we need a counter to tell the computer when the cycle gets to 50. We set the counter at zero at the outset by storing the number "0" at location 4. Now let's begin the cycle, which consists of adding the number "10" to the number "20." Incidentally, every piece of information is given a numerical address.

Let us assume that figure "10" has been stored in the computer in location 1, and that the figure "20" has been stored in location 2. We now need a flow chart as a basis for programming the solution.

The following rudimentary flow chart may give you the general idea.

FLOW CHART

```
STORE 0 IN L4
READ INFO IN L1
READ INFO IN L2

(Loop)
   L1 + L2 L3
   L4 + 1 L4
   L4 = 50? Yes

PRINT
STOP
```
The first instruction is to go to location 1 and pick up the number “10.” Next, the computer is instructed to go to location 2 and pick up the number “20.” It is then instructed to add (by counting) the number that it got from location 1 to the number that it got from location 2. The resulting number 30 is stored in location 3.

The next step is to trip the counter. The computer is accordingly instructed to go to location 4, which contains the number “0,” and add to it the number “1” and store the result in that location. The information in location 4 now reads “1” instead of “0.”

Next, the computer is instructed to take the number in location 4 and compare it with the number “50.” If it is not equal to that number, the computer is instructed to go back to the beginning and repeat the cycle. Repeating a cycle is designated by a “loop” on the flow chart. The computer adds 10 and 20 again, and this time the counter is tripped by adding a second number “1.” The resultant number “2” is then compared with the number “50,” and this goes on until the computer has completed the cycle 50 times. When location 4 contains a number that equals 50, the computer response, instead of being “No,” becomes “Yes,” and the computer is then instructed to print out the aggregate result of counting to 30 fifty times.

Although for the experts this is a trivial example, it gives some inkling of the kind of analysis that we have to go through when programming. Because for most lawyers the most important thing about a computer is its capabilities and limitations, it is desirable to know why a computer is programmed in such a way.

The computer's central principle is that, whereas you and I can count to 10 and far beyond, a computer can, in the first instance, count only to two. Why? Because it builds on a capability that permits only two basic conditions. A magnetized area is either positive or negative. A switch is either on or off.

The whole theory of computer operation is built around this fact. It means, first, that a computer can symbolize two, and only two, digits. It means also that in symbolizing alternatives there is no room for 5 choices, 4 choices, or even 3; a computer program must be couched in “two-valued” (either/or) logic, which makes it a sort of sophisticated game of “20 questions.” Fortunately, any problem can be analytically reduced to a series of choices each of which is to go this way or that way.

This is why the computer operates by exploiting the potentialities of the binary system. This is a system of notation that uses the base of 2 rather than the base of 10, which is used in the conventional decimal system. The notation is simple. The “on” position is designated by the symbol “1.” The “off” position is designated by the symbol “0.” The binary system works like the decimal system, except that instead of moving to the next column to the left every time you reach a multiple of 10, you do it every time you reach a multiple of 2.

Here is a chart that relates ordinary decimal digits to their respective binary equivalents.
<table>
<thead>
<tr>
<th>DECIMAL DIGIT</th>
<th>BINARY EQUIVALENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
</tr>
</tbody>
</table>

Those extra zeroes are there mainly to keep the binary counterpart of each conventional digit the same length. Notice that when you get to the binary equivalent of the digit “2,” it’s time to shift one place to the left. The same happens when you get to the equivalent of the digit “4.” You don’t shift again until you get to the equivalent of the digit “8.” The next time you shift is when you get to 16.

We could extend this table greatly, but most computer methods stop with the digit “9,” because the numbers 0 through 9 can be expressed in four binary digits. Thus, using the equivalence table for numbers 0 through 9, you can express the number “4798” by stating in succession the binary equivalent of each of the constituent digits. The respective binary equivalents of 4, 7, 9, and 8 may thus be combined as follows: 0100 0111 1001 1000.

This can get very complicated. It takes many locations and spots in a computer memory to deal with a relatively small item of information that in conventional notation we could write in a small space. For example, it takes 8 bits to store one letter of a word and 4 bits to store one digit of a number. This is another instance where the operations of the computer are less efficient than the comparable operations of human beings.

One value of computer programming is that the programmer cannot be a fuzzy thinker and succeed as a programmer. He has to understand in detail what he is doing. There is valuable mental feedback from this type of system. The same is true of framing specific legal questions for the computer. If the lawyer is to come up with usable results, he must do a detailed, logical, and coherent study of his problem. This may be hard on some lawyers but it is good for the law. Far from upsetting the law, a computer may have a valuable substantive therapeutic effect. It can’t cope with ambiguities or vaguenesses, but it can help to improve our deductive reasoning.

And so, we should not only be overwhelmed by computers, we should find them useful. The main thing is to find out what they can do and what they can’t do. In the law, computers can be useful in many instances in relieving the tedium and loss of time resulting from the types of routine operations that plague us every day as lawyers. At the same time, let us be rid of the nonsense that computers can draft instruments or do anything else that involves significant legal judgments.

When we add it all up, there is plenty of room in today’s legal world for both computers and lawyers. Let’s face the fact imaginatively and courageously.