Part 0

Introduction: The Software Age

Adapted from an article first published in
Sophisticated:The Magazine

1953 "Software" was merely a prank.

In October, 1953, I coined the word 'software.'

The notion of software as a separate thing from hardware took years to assert itself. Sure, the computer (popularly referred to as a "giant brain" in the early fifties) was unable to do anything but consume electrical power until a "programmer" came along to "program" it, and the consequent "routines" resided in the computer's "memory" thereafter. One did not, in the beginning, take a program written for one computer and put it into another. A half-century later, most people will find that hard to imagine.

As originally conceived, the word 'software' was merely an obvious way to distinguish a program from the computer itself. A program comprised sequences of written -- changeable -- instructions each endowed with the power to command the behavior of the permanently crafted machinery -- the "hardware."

For the origin of the word 'software,' most dictionaries give an unknown source and 1960 as the date, but [expletive deleted] I was there! That's the only exclamation point I intend to use in this introduction.

Fifty years ago, I was writing programs for the SWAC at UCLA, one of only 16 digital computers in the whole U.S. When I was first struck by the word 'software' (it was an epiphany), I shook my head and chuckled.

When I first said 'software' out loud, people around me said, "Huh?" From the very beginning I found the word too informal to write and often embarrassing to say. Nevertheless, with smirking trepidation I did occasionally feature the word 'software' in speeches and lectures and media interviews throughout the fifties.

It was just a throw-away thing. The word 'software' was hardly my most notable invention, even back then. Nothing to write home about (I was only 19 years old and still living at home). The word 'software' did not belong in a technical paper (besides, an undergraduate is but a ghostwriter for principal researchers). Then too, I had a reputation at UCLA as a practical joker.

Colleagues and friends simply shrugged, no doubt regarding each utterance as a tiresome prank or worse, another offbeat neologism, for which I was also becoming noted.

Nobody in 1953 would have guessed that the silly word would take hold, that within a few decades 'software' would enter the general vocabulary for products and for professions -- that a worldwide industry would wear it as a solemn name. You can be sure that if my ego and I had harbored any such glorious visions, then... then, what?

At the time of this writing, there are about 4.2 million issued patents (I once met Bob Hudson, inventor of an asphalt recycling machine for which he was awarded U.S. Patent No. 4,000,000 -- but I digress). There must be at least that many more worldwide -- a total, say, of 10 million. Every patent has some number of "claims" (my largest -- ahem -- for the first digital cassette recorder has 106 claims). Each patent-claim is, in effect, a separate invention, an average of perhaps ten per patent. So there must be at least a hundred million officially recognized inventions in the world. Punch Line: There are only 38,000 words in the English Language.

Nota bene, the hundred million issued patent-claims officially recognize only a small fraction of the world's inventions, but the rest are inventions just the same. An inventor does *not* apply for patent protection for each and every idea before reducing it to practice. Likewise, a person inventing a word does *not* submit it for publication in a dictionary before speaking it or writing it.

By comparison to other inventions, words are rare indeed. One reason is that there is no commercial value to them, which is why patents do not apply to words: You want to say "holomorph"? -- get out your checkbook. Copyrights don't make sense either, since a word uncopied has no linguistic value at all. Trademarks, truth be known, protect the interests of buyers not sellers; accordingly, a "wordmark" would benefit listeners and readers not speakers and writers. Words are *coined* and thus spent -- circulated, borrowed, defaced, worn out, and lost.

Dictionaries are not gate keepers like the Patent Office but merely infrequent periodicals, passively reporting on linguistic usage, relying necessarily on snippets in published documents -- in non-dictionaries. Once usage for an invented word becomes established, however, the etymological trail gets cold quickly, and if the original utterance is undocumented, even a linguistic bloodhound may not pick up the scent of the true inventor. Pity the verbal coiner, unrewarded, unacclaimed, unattested.

Inventors themselves, it seems plain, do a whole lot better job of inventing *things* than inventing words to call things. The bicycle got its 'pedal' from the piano, the engine appropriated 'piston' from the trombone, the airplane named its 'fuselage' from the sewing spindle and its 'empennage' from the tail feathers of an arrow. Millions of inventions apply to computers, both hardware and software. Readers are invited to check What's Not in a Name? and Mukashi for lamentations about the dearth of inventive language in 'The Software Age'.

'The Software Age' -- now, there's an ambitious coinage. The previous sentence deserves an exclamation point and so does the next one. Without software, there would be neither "The Computer Age" nor "The Information Age." Same for "The Space Age." Meanwhile, is "The Atomic Age" officially over? How much longer will "The Petroleum Age" last? Accordingly, does 'The Software Age' have any real competition?

All right, so one can get a patent on an invention, a copyright on a book, a trademark on a name, but what does one get to assert primacy as the originator of a word? As noted above, there is no proprietary instrument that protects a word. Nor, of course, would one want one ("I'll sue you if you use that term, I have the 'wordright' on it"). On the contrary, the idea is to put each coinage into currency, not horde it away.

And then the grandkids grow up and there comes a time to write one's memoirs. This one is dedicated to my first great-grandchild, Charlotte.

Part 1

Retrospective: Hardware on a Breadboard

In recent recent years, I have thought back to 1953 -- and beyond, to the 1940s -- probing my memory for the earliest relevant recollections. First to mind come all those electronic projects I worked on as a kid, including crystal sets and code-practice oscillators to earn Boy Scout merit badges. Later, I spent hours poring over plans clipped out of *Popular Science.* One summer, I built a superheterodyne short-wave receiver, using war-surplus tubes and transformers. At night, when atmospheric conditions were just right, I was able to pick up the BBC and other far away stations speaking strange languages.

My father was an electronic technician during World War II. He watched my juvenile struggles. One problem I had early on was to bring solder and iron and wire together -- with only two hands. He taught me the art of sculpting pre-tinned leads from twisted strands dipped in greasy flux and gently touched to the iron tip, thence to be drawn from the acrid smoke all silvery and ready for conductive attachment. My dad left me alone, except to grin later, with earphones pressed to his ears, as I tapped out C-Q ("seek you") on my home-made telegraph key.

My mother donated a cracked breadboard to serve as a substrate for my screwed-down treasures. That's right, a literal "breadboard," which by coincidence, let's assume, continues into the present century as the universal term-of-art for prototypes.

The workbench in our garage held stacks of coffee cans and shoe-boxes filled with screws and nuts, banana jacks and clip-leads, wire and terminal strips -- a vast inventory of parts purchased with my lawn-mowing wages at the neighborhood hardware store. That's right, "hardware" store -- decades before Radio Shack and Federated.

Algebra was my favorite subject in high school. My instructor noticed I was turning in all the problems at the end of each chapter, not just the odd numbered ones that he had assigned. Mr. Russell (not Bertrand, I am quite sure) lent me a book on symbolic logic, which contained a chapter on "Boolean Algebra."

There was something wonderfully definite about truth and falsity, about premises and conclusions. Soon the "syllogism" reached my eyes.

In a matter of days, I became fascinated by the "valid deductive argument" much like -- well, much like I became fascinated with girls later on. In 1949, I cleared my workbench and set about to breadboard my first invention, a "syllogism prover," using rotary switches and rejected relays from bombsights and autopilots brought home by my dad.

My inspiration was a paper by Claude E. Shannon published in the late 1930s on symbolic logic and summarized in a retrospective that appeared in *Science Digest*, to which I owned a subscription and read monthly from cover to cover. The first version of my syllogism prover operated on dry cells and had four main parts.

1. An eight position rotary switch was wired to a set of three relays. Each relay had multiple contacts capable of asserting and negating three Boolean variables X, Y, Z.
2. A patch panel enabled the contacts of the relays to be wired in series and parallel to provide the respective Boolean operations of ANDing and ORing.
3. Two relay solenoids were wired to the patch panel to receive logical signals such that their normally open contacts representing "premises" P1 and P2, which were wired in series, producing a TRUE signal only when both premises were TRUE.
4. A third relay solenoid was wired to the patch panel to receive logical signals, and its normally closed contacts representing that the "conclusion" C was FALSE, which was wired to a lightbulb labeled "Invalid."

It might be noted in passing that the syllogism prover was by definition an "analog" computer, inasmuch as individual parts of the assemblage corresponded to specific parts of the problem being solved (a different problem required a different configuration of the physical components -- the "hardware"). That's the proper definition of an analog computer -- never mind the fact that the variables and operations inside were limited to two digital values, TRUE and FALSE. In today's usage, "analog" has become synonymous with "continuous," "digital" has come to mean "discrete," and, thanks to advances in digital technology including software, "analog computers" have all but disappeared from the face of the planet.

There were no science fairs at my school, which is just as well. The syllogism prover took hours to set up (to "program" in the parlance of computers). The contraption would then be seen operated by a bespectacled nerd clicking the rotary switch and gazing at a small lightbulb on the panel, which for a "valid deductive argument" was supposed to stay extinguished. A real yawner. Even my dad shrugged when he saw it. Soon the syllogism prover was cannibalized to build a tick-tack-toe machine, which I never completed.

Thus, as far back as the late forties, the term "hardware" came naturally to me as I pedaled my bicycle home from the neighborhood hardware store with a sack of components for my adolescent projects. Moreover, in 1947, according to the *Oxford English Dictionary*, the word "hardware" was already being applied to computers as a synecdoche.

Today, we hear the expression "military hardware," which, for all I know, is gratuitous slang, an informal expression used to distinguish capital equipment from personnel and consumable supplies. Manufacturers might do the same with "factory hardware," but they don't. Neither do construction contractors ("excavation hardware") nor cosmetologists ("hairdressing hardware").

Many people harbor the suspicion that for computer systems, the word "hardware" arose as a retrospective designation -- nomenclature made necessary in the sixties *after* the word 'software' became current. If so, then "hardware" would be characterized as a "retronym," much like the word "ordinary"-- a term which became necessary to distinguish the high-wheel bicycles of the 1880s from those new-fangled "safety" bicycles with their chains and sprockets that became popular in the 1890s.

The practice of retroactive naming occurs in other realms ("World War I" replaced "The Great War" while "World War II" was raging in Europe) but not all (there is no film entitled "Rocky I").

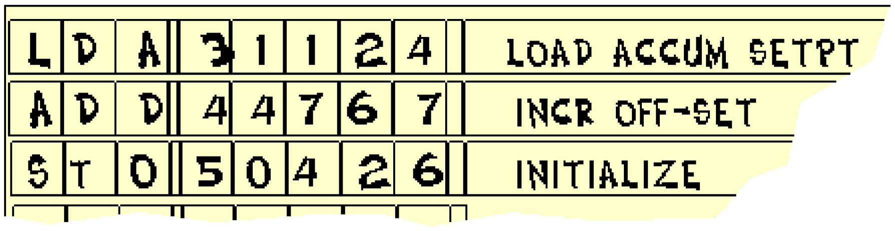
However, as a term-of-art in computers, the word "hardware" was not a retronym, as confirmed by citations in the OED.   An adolescent subscriber to *Science Digest* might well have applied that terminology.

Now, I am not saying that the syllogism prover in 1949 sensitized me to the distinction between 'hardware' and 'software.' That came later -- four years later. Meanwhile, I had soldered up a *permanently crafted* set of resources, comprising battery, breadboard, and relays. Somewhat separately, the machine was equipped with a rat's nest of *changeable* logic in the form of jumpers on a plug-board (much like the earliest EAMs -- electrical accounting machines, come to think of it).

The mere fact that the logic could be changed did not seem to me in any sense "soft," even though it was far more convenient to go about foozling with jumpers and sockets than unsoldering and resoldering tinned wires on terminal strips. No, I am confident in the recollection that the mental precursor for "softness" necessarily waited for personal experiences with another kind of changeability.

Four years later, I won a scholarship and entered UCLA's College of Engineering as a junior. In June of 1953, I landed a dream job on campus at the Institute for Transportation and Traffic Engineering (ITTE). That marked the beginning of two years filled with memorable experiences.

During the first week at ITTE, I met the SWAC.



```
LDA 31124  LOAD ACCUM SETPT
ADD 44767  INCR OFF-SET
ST0 50426  INITIALIZE
```

Sample of a program coding sheet, circa 1953 (sketched from memory).

Part 2

Epiphany: Dan Gerlough and the SWAC

The toll for a telephone call from Redondo Beach to Westwood would have been more than I could afford, so I did not return Dan Gerlough's call. Eventually he caught me at home.

"Your title will be 'research assistant'," said he, "and the position pays $1.57 per hour." I was already making $1.75 at a service station over on Pacific Coast Highway, and I told him so. Gerlough continued in a deliberate voice as if I had not spoken. "The work here at ITTE [Institute for Transportation and Traffic Engineering] includes instrumentation for automobile crash-injury experiments, human factors studies; you will have your own office and laboratory here on campus; there will be some computer programming on the SWAC, and -- "

"When can I start?" I asked.

While outfitting ITTE's new electronics laboratory a few weeks later, I was assigned to write programs for the SWAC, which was located in the mathematics building near UCLA's Royce Hall. I had studied the programming concepts on my own, drenching my mind in how a "stored program digital computer" operates, how programs in the computer's memory could run in branching pathways and in loops, how they could modify their own instructions each time around, processing tables of data and making decisions on the fly. I became captivated by the realization that instructions -- my instructions -- would be executed by the thousands every second. I had eagerly devoured the SWAC manual, learning its command set and the procedures for getting programs prepared and loaded and run.

The SWAC filled a room the size of a tennis court. It occupied row upon row of cabinets with cooling fans humming, each cabinet containing stacks of Williams tubes for memory and mercury delay lines for registers. A galloping punch-card machine appropriated from the realm of EAM (electronic accounting machines) was used for inputting both data and programs; a clattering line printer nearby was the output. There was a glass wall through which visitors to the campus, including school children and conventioneers -- once even a whole convent of nuns -- would stand wide-eyed, sucking their lower lips and shaking their heads before this marvel of mid-twentieth century technology, while a moustachioed docent in cardigan and necktie explained the significance of what they were seeing just beyond the window: A giant brain.

Dan Gerlough was doing research for a paper on vehicular flow, and I was allowed to spend more paid hours on the project than I could stay awake. I made plenty of errors in my earliest coded programs, of course. The term "bug" had already been appropriated for both mistakes in programs and hardware failures. It was not always possible to tell which to blame, because the SWAC went down at least once a day. Don't tell anyone, but I would rest my head on the keypunch machine and doze sometimes for hours waiting for the computer to be repaired.

The university employed a gray-haired PhD for maintenance of the SWAC. I cannot remember what I had for breakfast today, but I shall never forget the sight of Dr. Whitcomb running out of the repair shop holding up a window frame festooned with "bottles" (vacuum tubes) and multicolored componentry. "Aha!" he exclaimed. "It was the epsilon flip-flop."

Dan's deadline was fast approaching. My subroutines for one simulation program had to be perfect; I had checked them a dozen times. Dan agreed after looked over my coding sheets. "It's that machine," I complained. "It's broken down again." Electronic though it was, from the beginning a computer was called a "machine." I came to resent the machine. More than once I told Dan, "I'm glad I don't have Dr. Whitcomb's job."

Surely, I was thinking to myself that I wanted nothing to do with the SWAC "hardware" -- that the machine was the mindless means for executing my programs -- a necessary evil, mostly evil. It was about at that moment, I seized upon the consummate reality of what I was doing -- that what I was doing was sharply different from what Dr. Whitcomb was doing -- that what I was doing was writing on a coding sheet, not plugging jacks into sockets, not clipping leads onto terminal posts, not soldering wires, not bending relay contacts, not replacing vacuum tubes. What I was doing was writing on a coding sheet! The exclamation point was right there in my thought back then and in my memory now.

It was October 1953 and I was experiencing an epiphany. Before my eyes, I saw my own markings carefully scrawled inside printed blocks on the coding sheet. They comprised numerical "words" -- the only vocabulary the computer could understand. My coded words were not anything like those other things -- those machine things, those "hardware" things. I could write down numerical words -- right or wrong -- and after they were were punched into cards and fed into the reader, the SWAC would be commanded to perform my mandated operations in exactly the sequence I had written them -- right or wrong.

The written codes -- *my* written codes -- had absolute power over Dr. Whitcomb's "hardware." Then too, I could erase what I had written down and write down something different, then punch a new card and insert it into the deck. The SWAC, slavishly obedient in its hardware ways, would then be commanded to do my work differently -- to do different work entirely, in fact. The writing on the coding sheet was changeable; it was decidedly not hardware. It was -- well, it was "soft-ware."

Dan Gerlough frowned. He had a far-off look in his eyes when I said the word, then chuckled somewhat dismissively when I explained its meaning. I could tell he was worried about meeting his publication date. Dan hefted his notebooks and turned to leave. "Paul, stay here and be ready to run your -- your 'soft-ware,' just as soon as Dr. Whitcomb says the machine is working again."

During the two dozen months that followed, in between constructing a catalog full of experimental circuitry for the landmark studies going on at ITTE, I had many occasions to use my new word, first among the ITTE staff and later in the talks about "giant brains" that I gave at high-school science classes and career days and in speeches to service clubs all over Los Angeles.

Jocularity does not always sit well among listeners who are struggling to comprehend new concepts. "We may yet wind up on the inside of a zoo," I would say (and still do), "with computers on the outside." Yes, I could be something of a wag, all right. *The Daily Breeze* quoted me as saying, "Most people regard the computer as combining the attributes of a sorcerer and a mad -- but competent -- accountant." A few people in my audiences would allow themselves to laugh.

The word 'software' had a more predictable effect, producing quizzical expressions and shrugs. "Without its 'software'," I enjoyed saying, "a giant brain is less intelligent than a giant gall bladder." Of course, I always hastened to explain what 'software' is.

The word "programming" was serviceable enough, of course. Renaming it "software" did more to assure my reputation as an eccentric than to illuminate computer technology.

Ah, but when you're there, an invention is wonderful! -- the moment when a new idea arrives is its own exclamation point. Even more so, when the invention is a new word.

Dr. Gerlough more or less liked the term 'software.' More *less* than more. Others at ITTE -- John ("Harry") Matthewson, Slade Hulbert, Bob Brenner, and Paul Barber -- were polite enough to be amused. My colleagues at UCLA solemnly kidded me: "Don't let Dean Boelter catch you over there at Royce Hall playing with your software." They viewed my latest neologism as a good-natured prank, I think.

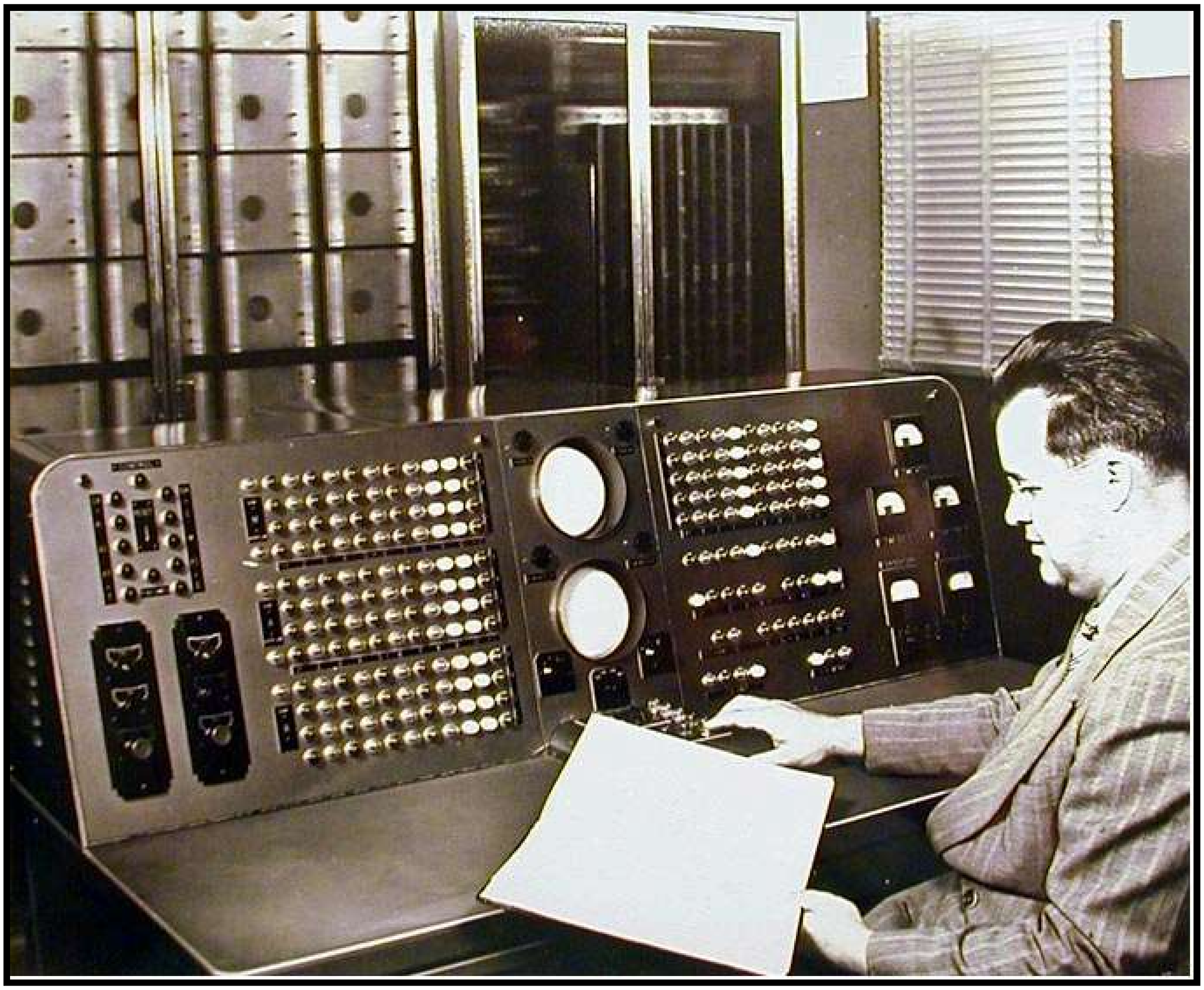
Who would have expected the silly word ever to catch on? To be spoken with a straight face? To become the name for a profoundly influential industry? In researching its provenance, I have been deeply saddened to learn that nearly all of my colleagues at ITTE, including Dan Gerlough, have passed away. I will always be grateful for Dan's persistence in hiring me away from that service station on Pacific Coast Highway. And for giving me an opportunity to write *soft*-ware for the SWAC.

Try to imagine, Dan, that hokey word 'software' has actually been taken seriously!

The SWAC

In 1953, there were few computers indeed in the U.S. and Europe. Here is a representative list with their respective years of introduction.

- 1940 Bell Labs Model 1
- 1941 Iowa State Collage ABC (Atanasoff-Berry Computer)
- 1943 Harvard ASCC Mark I
- 1945 University of Pennsylvania ENIAC (Electronic Numerator, Integrator, Analyzer, and Computer)
- 1945 University of Pennsylvania EDVAC (Electronic Discrete Variable Automatic Computer)
- 1948 Harvard Mark II
- 1948 Manchester University Mark I
- 1949 IBM 604
- 1949 MIT Whirlwind
- 1949 Cambridge University EDSAC (Electronic Delay Storage Automatic Computer)
- 1949 U.S. Air Force BINAC (Binary Automatic Computer)
- 1950 Harvard Mark III
- 1950 National Physical Laboratory ACE (Automatic Computing Engine)
- 1951 US National Bureau of Standards SEAC (Standards Eastern Automatic Computer)
- 1951 Remington Rand UNIVAC (Universal Automatic Computer)
- 1952 NBS **SWAC** (Standards Western Automatic Computer)



Standards Western Automatic Computer (SWAC)

Photo from *The History of Computing: An Encyclopedia of the People and Machines that Made Computer History*

With an add-time of 64 microseconds, the SWAC was the fastest stored program digital computer for its time. Known as the ZEPHYR, the machine was developed by the National Bureau of Standards Institute for Numerical Analysis at the University of California at Los Angeles. Start dates included development in 1948, construction in 1949, and operation in 1950, continuing in service for 17 years and finally retired in 1967.

The SWAC differed from the NBS's other main computer the SEAC, in that the SWAC operated in a parallel fashion rather than serially. The SWAC used 37 Williams-tube memory units; magnetic drum storage was added in 1953. Some of those who worked on the SWAC project and its later enhancements included: Harry D. Huskey, H. Larson, R. Thorensen, M. Melankoff, D. Lehmer. {*Return*}

<< **Previous** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **Next Part** >>

Part 3

Tribute: John von Neumann

Man from Budapest

adapted from from an article in
Sophisticated: The Magazine

The road wound through the hills of Pennsylvania. I steered as close to the line as I dared. The tires on the rented stationwagon squealed, making my passengers gasp. All three of them must be wondering if they have made the right decision to share expenses with me. I glanced at my watch and shook my head. Never make it.

The gentleman in the seat beside me gaped through the windshield. He asked politely about the paper I gave yesterday at Pennsylvania State University. That was several miles ago. "Computer-driven displays," I had answered. I have to concentrate on the road. If I miss the last flight out of Pittsburgh, I won't get to my home in Somers Point, New Jersey until Saturday afternoon.

"It was most informative," said a man in the seat directly behind me. I glanced in the rear-view mirror. He craned his neck to catch my eye and smiled. The man, who had introduced himself as Ted, was apparently the only person in the car who had attended the AIEE (American Institute of Electrical Engineers) conference where I gave my paper.

"It was the second version," said I, unskilled at receiving compliments with modesty. "My first, which was delivered at a 'Track-While-Scan' conference in Boston, has been quite popular."

This frantic automobile trip was made necessary by high winds at the College Park airport. Standing outside the terminal building, the four of us had watched the twin-engine Martinliner 404 being tossed about in the sky, circling the field. An announcement crackled over the public address system, "Flight [whatever] has been cancelled."

My passengers are professors and administrators at Penn State. In this group, I was the only visitor. That gave me a minor celebrity status. We all had one thing in common: a need to make connections at the Pittsburgh airport, which is a hundred miles to the west of State College. I was in the greatest hurry -- and my destination was in the opposite direction. Philadelphia International is a hundred miles to the east.

"You are only the second person I've met who actually works with computers," said the gentleman beside me. That was not an extraordinary statement in 1959. The man had introduced himself at the car rental. "My friends call me Zach," he said. Zach was about the age of my father, distinguished in his grey vest and silver hair. He did not use the expression "giant brains," which would have afforded me an opportunity to disabuse him and my other passengers of a common myth of that time.

"My first computer was the SWAC at UCLA," I said.

"Let me out of the car," protested Ted. The man was in his thirties and wore a flat-top, obviously a USC fan.

"SWAC stands for 'Standards Western Automatic Computer'," said I. "Also called the 'Zephyr', the SWAC was the fastest computer for its time." I explained that in the earliest days, each computer had a name, most commonly ending in '-AC' (for automatic computer): EDSAC, ENIAC, UNIVAC. By the late fifties, computers began to have model numbers. For my research in air traffic control, which I had described in my paper, I was programming the IBM 704 and building what are called "real-time" systems using the computer sold by my company, which was the RW-300.

"Who invented the computer?" asked the psychology professor from the back seat, the only woman in our party.

"Many persons, certainly," I replied. The road into Altoona became straighter, and I like to talk. I glanced over my shoulder. The woman was in her forties, which for me at that time meant old, with graying hair pulled back in black combs. I commenced an excerpt from one of my lectures at Los Angeles Tech. "Any list of inventors will have to include four names: Babbage and Boole in the last century, Turing and von Neumann in this century. My idol since childhood is John von Neumann. I should say, he was my idol. He died last February. Anyway, back in the forties, von Neumann wrote a paper -- "

"Did you know him?" Zach asked suddenly. There was more than curiosity in his interruption. I wondered what it was.

"A couple of years ago, von Neumann visited the RAND Corporation in California. I had an opportunity to hear him speak. He accomplished so many things: statistical mathematics, quantum physics, game theory -- you know, 'saddle-points': where 'minimax' meets 'maximin.' Then in computers, he -- "

"So you at least met Johnny," Zach said, with a strange insistence in his voice.

I stole a glance. "You call him *Johnny*?"

Zach nodded solemnly.

"Hey, you must have *known* him," I exclaimed.

He nodded again and looked away. I focussed on the road ahead. I wanted to find out more, but before I could pose a question, Ted spoke up. "What was the paper about?"

"Von Neumann described a new machine, which was eventually built -- the 'EDVAC' -- 'Electronic Discrete Variable Automatic Computer.' His paper was the first description of a stored-program computer. It will be known for all time as 'the von Neumann machine'."

"What is so special about the von Neumann computer?" the woman asked.

"The short answer is, it executes programs made up of modifiable commands. The programs are held in a 'store.' Some people like to use the word 'memory,' but it's not much like the memories we have. Anyway, stored-program computers perform operations on what we call operands, and -- this may sound crazy -- they perform operations on their own instructions! That's what's special about them as much as anything else. That and making decisions based on the values of operands; they do that, too, thanks to John von Neumann. Imagine, he even had a computer named after himself: the JOHNIAC. I have been reading about his work since high school. Once you have programs that can change themselves, and -- oh yes, speed -- well, there's no limit to what you can do."

That exchange launched a lecture to my captive audience that continued all the way to Pittsburgh, delving deeper into the early ideas of computers and the amazing feats of von Neumann. "He was my DiMaggio," I said at one point. "There's this famous story about the first hydrogen bomb," I said, "which was soon to be set off in the Pacific Ocean." I explained that one out of every 2,500 water molecules has a deuterium atom instead of hydrogen. "'Heavy water,' it's called. Deuterium is just what you need to make a hydrogen bomb. The question was: Will an H-bomb set off a chain reaction in the ocean itself and destroy the earth?"

"Kind of an important question," mused Ted from the back seat.

"And the answer, yes or no, required a huge amount of statistical calculations, for which the Atomic Energy Commission commissioned a new computer. John von Neuman was brought in to consult on the project, and during one meeting, he was seen scratching out some calculations. In minutes, von Neuman presented the solution and then asked, 'What do you need a computer for?' Someone laughed. 'Because we don't always have you, Johnny'."

With no prompting from my passengers whatsoever, I began talking about my father, who filled our garage with surplus electronic components after the war. In high school I was given a book on symbolic logic and decided to build a machine to test syllogisms. The contraption, which was built on my mother's discarded breadboard, had a truth-table generator and a plug-board on an unpainted plywood panel. I had plenty of relays and wire, selenium rectifiers and transformers, but I had to spend all my money at the hardware store for nuts and bolts and brackets. I could plug together simple "conclusions" and test them against logical "premises." A valid syllogism means that if the premises are true, the conclusion cannot be false. The syllogism prover occasionally worked: clicking along, "exhausting all combinations of truth and falsity upon the variables." I would watch the light bulb labeled "invalid" to see if it ever flickered.

My father puffed his pipe. "Quite a light switch you have there."

The road approaching Pittsburgh curved through verdant valleys. "To answer your question, Zach," I continued, "I never met von Neumann, but in junior college, I read every article about him in *Science Digest*. I studied everything I could find in the library on the stored program digital computer. I taught myself to program simple machines of my own design. A year later, in 1953, I met the SWAC at UCLA."

"Red Sanders had a great year in '53," said Ted.

I shrugged.

"Your coach at UCLA -- Red Sanders."

"Didn't have much time for sports," I said. "Working alongside my mentor, Dan Gerlough, I got my excitement mostly by writing 'routines' and 'subroutines,' 'loops' and 'branches'."

"The Bruins shut out the Trojans 13-0 and won the Rose Bowl."

"If you say so." I grinned at my passengers in the rear-view mirror. "Ah, but it really is exciting when you have a machine executing thousands of instructions every second -- tirelessly and exactly doing your work like magic. You prepare programs in special codes that the machine can read and understand and then you let it fly! That's the exciting part. When it works. Now, digital computers will make mistakes just as fast, too. Your mistakes. You have to fix your programs sometimes, but modifications are easy. You just punch up a Hollerith card and read it into the computer. Not like messing with those tangled wires on a plug-board. That was hardware. Programming is something else. Sure, you need hardware to run your program. Hardware is worthless without its programs. The hardware needs -- well, what I call '*soft*-ware'."

"I think I heard you mention 'software' in your talk yesterday," said Ted. "I wondered what you meant by that term."

"Software," mused the woman with the combs. "What a wonderful word!"

I glanced across at the man in the seat beside me. "Do you think John von Neumann would have liked the word 'software'?" I asked. Zach nodded unsmiling.

As we pulled into the Pittsburgh airport, I prepared to jump out and run for my plane. Zach had not spoken since before Altoona. I reached across to shake his hand and saw that he had tears in his eyes. His voice wavered. "My wife and I..."

Ted thanked me for the ride and offered to turn in the stationwagon. I opened the door and tossed him the keys.

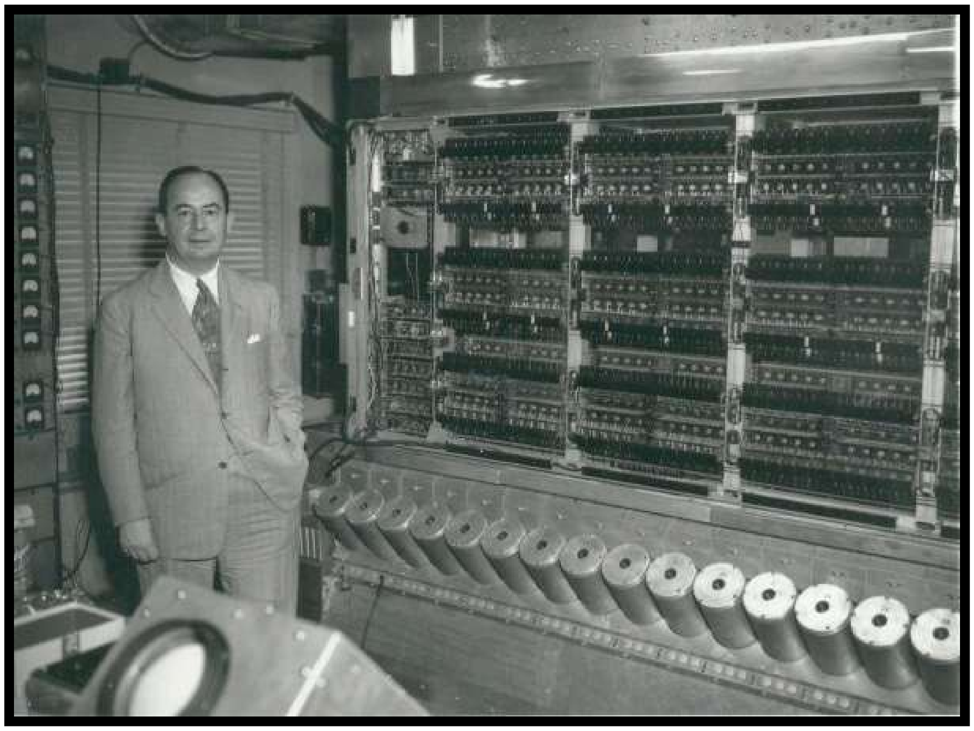
"My wife and I," said Zach, "were Johnny's sponsors when he first arrived from Germany. He and his wife lived with us for their first weeks in the U.S. She -- my wife -- will be touched when I tell her about you. And, from what you have told me this afternoon, I now have for the first time an understanding of Johnny's work, his contributions."

<< **Previous**	**0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**	**Next Part** >>
Home Page	**Softword: Table of Contents**	**Top of Article**

John von Neumann

The fifties marked the beginning of what I call '**The Software Age**'. Today, every engineer I interview must answer the following question: "What are the three greatest innovations in computers since von Neumann?" Most people say software. A few mention transistors or semiconductors. There are no wrong answers.

My favorite right answers are index register, priority interrupt, and the stack. You will notice, I did not include the cache memory (see Mukashi). Lately, in reply, I have heard the question, "Who is von... uh, who?"



John von Neumann (original name Johann) was born December 3, 1903 in Budapest, Hungary. He died of cancer on February 8, 1957 inWashington, D.C. He is described in *Encyclopedia Britannica* as follows: "Hungarian-born German-American mathematician who made important contributions in quantum physics, logic, meteorology, and computer science. His theory of games had a significant influence upon economics."

In 1926, von Neumann received a diploma in chemical engineering from Technische Hochschule in Zürich and the same year, he wrote his dissertation about set theory and received a Ph.D. in mathematics from the University of Budapest. He lectured at University of Berlin in 1926 through 1929 and at the University of Hamburg in 1929 and 1930. During this time he worked mainly on quantum physics and operator theory, which, because of his work, are now viewed as two aspects of the same subject.

In 1930 von Neumann was visiting lecturer at Princeton University; he was appointed professor in 1931; in 1932 he published important work on statistical mathematics and a book on quantum mechanics, *The Mathematical Foundations of Quantum Mechanics*, which continues as a standard presentation of the subject; in 1933 he became a professor at the newly founded Institute for Advanced Study and kept that position for the rest of his life.

In the second half of the 1930s von Neumann's publications included work on what has become known as "Neumann algebras," one of the most powerful tools in the study of quantum physics and from which emerged "continuous geometry."

According to the *Britannica*, "About 20 of von Neumann's 150 papers are in physics; the rest are distributed more or less evenly among pure mathematics (mainly set theory, logic, topological group, measure theory, ergodic theory, operator theory, and continuous geometry) and applied mathematics (statistics, numerical analysis, shock waves, flow problems, hydrodynamics, aerodynamics, ballistics, problems of detonation, meteorology, and two nonclassical aspects of applied mathematics, games and computers). His publications show a break from pure to applied research around 1940. During World War II, he was much in demand as a consultant to the armed forces and to civilian agencies. His two main contributions were his espousal of the implosion method for bringing nuclear fuel to explosion and his participation in the development of the hydrogen bomb."

As mentioned above, von Neumann's theory of games was based largely on the "minimax theorem," which he stated in 1928 and later elaborated with another of his sponsors, Oskar Morgenstern, publishing in 1944 *Theory of Games and Economic Behavior*.

In computer theory, von Neumann did much of the pioneering work in logical design, in the problem of obtaining reliable answers from a machine with unreliable components, the function of "memory," machine imitation of "randomness," and the problem of constructing automata that can reproduce their own kind.

What was the secret of John von Neumann's success? Many experts will answer, "the axiomatic method," by which he got to the root of the matter by concentrating on the basic properties (axioms) from which all else follows. His axiomatization has left a permanent mark on the subject; moreover, according to those who worked alongside von Neumann, his insights were illuminating and his statements precise. {*Return*}

<< **Previous**	**0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**	**Next Part** >>
Home Page	**Softword: Table of Contents**	**Top of Article**

Part 4

Claim: An Adventure Begins

Forty years later, in 1999, I was sipping cocktails at an Equinox party in the hillside home of my best friend, the late Tom Sullivan. A software enthusiast and computer historian, Tom curated a magnificent livingroom display of venerable computer technologies -- cores and disks, printed circuit boards and chips.

Tom introduced me to his guests. "Paul here is the guy who invented the word 'software'."

Nothing but shrugs from around the room. I felt my face fluoresce. Fully conflicted, with a familiar battle raging inside my cranium between modesty and pride, I studied the tops of my shoes. Not that I am shy -- on the contrary, I enjoy giving speeches.

> The typical program chairman will read his prepared introduction to the audience, stumbling over the words that describe "the inventions of this evening's speaker." A university audience will good-naturedly groan about "the radar speed meter"; an aviation audience will nod approvingly when hearing about "the altitude reporting transponder"; computer science audiences exchange glances when they are told that they will be addressed by the inventor of "the cache memory"; a school assembly will be awestruck by the presence at the podium of the owner and rider of "the largest bicycle in the world."

At a cocktail party, however, each person seems intent upon his or her own relevancies, indifferent to those of others, resisting the impulse to admire, taking refuge in a private rampart of doubt. Such occasions do not constitute the best venue for advertising achievements, whether bygone or recent. People are too polite, though, to put their skepticism into words ("Oh really? I suppose you have plenty of documentary evidence for that allegation.").

On this occasion, I sensed dismissiveness in the air. I gestured toward a huge dictionary that Tom keeps open on a pedestal across the room and shrugged. "You can look it up," I said, secure in the knowledge that nobody would bother to do so. Despite its thin Bible-paper pages, the volume stands fully eight inches tall and contains an entry which reads...

> ***soft wares*** = Dry goods. Dry goods Commercial -- Chiefly U.S. Textile fabrics, cottons, woolens, linens, silks, laces, etc. -- in distinction from hardware, jewelry, groceries, etc.
>
> -- *Websters New International Dictionary, Second Edition, Unabridged* 1954

Now, there are plenty of media references on the first radar speed meter and published papers about the altitude reporting transponder. Patent 3,938,097 for the cache memory hangs on my office. Hey, I even have my picture in the *Guinness Book of World Records* astride The 64-inch Columbia Expert Ordinary. Nevertheless, in 1999 there existed not one dictionary that associated my name with the word 'software'.

As the Centennial -- yea Millennium -- rang in, I became absorbed in the thought that the time had arrived for me to deal with certain matters of life-relevance. Top of my list was the authentication of my claim for the word 'software'. It should be easy, I thought.

It was not.

Then and there began a private effort that spans six years of evenings and weekends. What I like to call the "softword project" has been an adventure full of suspense and with a surprising outcome -- outcomes, plural.

Starting out, of course, I consulted dictionaries and other references to ascertain what is known about the origin of the word 'software'. Dictionaries cite an earliest date of 1960. The one that really counts most is the *Oxford English Dictionary*, so I took out a second mortgage on my house and bought the 1999 edition. Appendix A provides the complete OED entry for 'software' along with its historical citations. Here are salients...

Sense *a.* The programs and procedures required to enable a computer to perform a specific task, as opposed to the physical components of the system.

Sense *b.* The body of system programs, including compilers and library routines, required for the operation of a particular computer and often provided by the manufacturer, as opposed to program material provided by a user for a specific task.

Given the scholarship that characterizes every entry in *OED*, and given the significance of the word 'software' -- indeed 'software' is a word offering itself for the naming of the present *age*! -- perhaps a little parsing will be tolerated here.

Of crucial significance is the fact that Sense *a* empowers software "to perform a *specific task*," while drawing the distinction enjoyed by software with respect to "*physical components*" -- hardware -- the latter commonly regarded as "general purpose." Sense *b* rightly depicts "system programs" as implements of the computer art, wherein machines have increasingly participated in their own programming. By the way, non-computer senses for the word 'software' -- books and films and videos -- are addressed elsewhere.

The *OED* entry expands on the definition as follows: "In early use, the word was interpreted widely to include program material written by a user, as well as systems programs...." Quite so -- but only for tardy values of "early." In its earli*est* uses, the word 'software' referred almost exclusively to "material written by a user." My claim not withstanding, the term would almost never have been interpreted to include "systems programs," since there were few of the latter prior to 1960 and they would have been primitive indeed. For every application of a given computer in those early days, the user necessarily created the whole program -- and in machine language, at that. The first such was binary. Still is. Lurking deep inside today's most advanced computer chips are still those primordial binary bits of old, the *intrinsic* machine language. In the late fifties, various "utility packages" were being developed comprising globally accessible subroutines for interfacing to the hardware for read/write functions, for sensing, bootstrapping, loading. These support services were of lesser importance than "applications" programs and in the early sixties would be absorbed inside the earliest "monitors" and "operating systems."

In the fifties, software was not in the modern sense "written." Software needed to be punched into paper tape or cards. For fixing "bugs," a programmer might even be required to operate rows of toggle switches and push-buttons on a panel to input "instructions." Each instruction comprised an "operation code" (load, add, subtract,...store) alongside an "operand address" -- all in binary. Early progress was marked by allowing clusters of binary bits to be encoded as octal digits -- a forlorn convenience. Thus the earliest software was necessarily expressed in what became pejorated as a "computer oriented language" (COL), wherein the properties of the hardware were always starkly in evidence -- registers, operations, memory addresses.

Any COL is the most *un*natural form of expression if you happen to be a human being. The earliest "assemblers" came along to make COLs more readable and mnemonic, but the programmer was still required to translate the requirements of each mission, whether a scientific calculation or a business accounting, into the machine's unforgiving COL. Along came "compilers," each of which accepted a "problem oriented language" (POL) as "source code" and automatically translated POL strings into the requisite COL, known as "object code." POLs and many other kinds of software belong in the realm of *tools*. Tools are not *jobs*, and the ultimate work of the hardware must be controlled by software -- programs and procedures -- that enable a computer to perform specific jobs.

The Cobol compiler, which is mentioned in the *OED* 1961 citation, got its name from an acronym for COmmon Business Oriented Language. In the sixties, COBOL was a boon for coercing computers to do general ledger, receivables, payables, payroll, labor distribution, labor performance, inventory control -- the real *jobs* we needed machines for. Before Cobol there was a POL named Fortran (FORmula TRANslator) and before that, in 1959, there was SOAP (Symbolic Optimizing Assembly Program), for its time the most advanced form of a COL.

Despite the perceptions in many references, the use of the term did *not* wait until assemblers and compilers came into existence. Software created directly in machine language was there from the git-go mandating each job's specificity and putting the usefulness into the general-purpose hardware.

The *earliest* quotation cited in *OED* is from the **June 1960** issue of *Communications, Assocation of Computing Machinery,* "Nearly every manufacturer is claiming compatibility with all other equipment via such software as Cobol."

> *Nota bene*, the off-hand expression "...such software as..." discloses evidence (a) that in 1960, the word applied to other classes of software (non-Cobol-like, let's say),

which would surely include application-specific, do-the-*job* programs and (b) that the word 'software' was in use before June 1960.

The second citation is a **June 1961** *Computer Bull* article, "The programming expertise, or ‘software’, that is at the disposal of the computer user comprises expert advice on all matters of machine code programming, comprehensive libraries of subroutines for all purposes, and the pegasus/sirius scientific autocode."

Here is evidence that in the 1960s, authors in the computer field were already becoming preoccupied by the magic of "systems programs," which, however attractive as programing *tools*, merely support the *preparation* of software and must eventually be trumped by the *execution* of software -- the ultimate *purpose* for software -- to do *jobs*.

There being no documented usages in *OED* of the word 'software' earlier than 1960, I wrote to the Chief Editor...

Subject: The Word 'Software'
Date: Thu, 06 Jul 2000 13:18:40 -0700
From: Paul Niquette <paul@niquette.com>
To: John Simpson <john.simpson@kellogg.oxford.ac.uk>

Dear John Simpson,

In October of 1953, I coined the word 'software.' No kidding.

Most dictionaries give an unknown source and 1960 as the date, but [expletive deleted], I was there! That's the only exclamation point I intend to use all week.

You people at OED will demand proof, of course, and I am pulling together some documentation. I have identified 28 likely witnesses in the 1953 time frame, but I have not yet contacted them on this subject, with the intention of not inflencing their respective memories. I have prepared a chronology, and I have drafted a memoir entitled, "Softword," which I will publish on my website.

In briefest summary, I was writing programs on the SWAC in 1953 at UCLA. At the time, the SWAC was one of about a dozen computers in the whole U.S. When I first hoked up the word -- to distinguish programs from hardware, of course -- people said, "Huh?" I explained that 'soft' meant changeable.   People then and for years later kind of sneered "Software...[pause]...I see."

The notion of software as a separate thing from hardware took years to assert itself. Sure, the computer (popularly referred to as a "giant brain") was unable to do anything until a 'programmer' came along to 'program' it, and the consequent 'routines' and 'subroutines' resided in the computer thereafter. One did not, in the beginning, think of taking a program written for one computer and put it into another computer. Remember IBM after the consent decree? "Unbundling" was the result, and software became a commercially distinct product line. But that was not until the sixties.

From the very beginning I did not find the word 'software' to be useful. It was embarrassing to say and too informal to write. I stopped liking the word. I saw -- excuse the immodesty -- I foresaw the incomparable hardness of software, as celebrated in my rather influential screed at...

http://niquette.com/paul/issue/softwr02.htm

...but with smirking trepidation I did go ahead and use the term from time to time in speeches and classes and media interviews throughout the fifties, before I got bored with it. I never expected the silly word to catch on. Later on, I started hearing and reading the word, generally accompanied by blushing explanations.

One can get a patent on an invention (I have 38), a copyright on a book (I have five), a trademark on a name, (I have several), but what does one do to assert primacy as the originator of a word. The idea is to put it into currency, of course, not horde it in some way. And then grandkids come along and there comes a time to write one's memoirs...

That was supposed to be a brief summary, Dr. Simpson. What do I do now?

Best regards,
Paul

From: John Simpson <john.simpson@kellogg.ox.ac.uk>
Sent: 06 July 2000 23:33
To: Paul Niquette <paul@niquette.com>
Subject: Re: The Word 'Software'

Dear Paul,

Thanks for your note about "software".

You're right that we'd be very interested in (preferably published) documentation from the fifties in which you use the word. It's a long shot that you'll have any unpublished documentation from that time, but maybe you do!

Sometimes there are established "stories" about the invention of a word which we might refer to (again preferably in published sources) if actual documentation is missing.

You'll understand the problem. We may have a number of people all claiming to be the inventor, and without any evidence it's not possible to decide between competing claims.

But please do let us know what you can dig up.

John Simpson
Chief Editor, OED

Thus advised, in the summer of 2000, I took up an action plan, focusing on three searches, for...

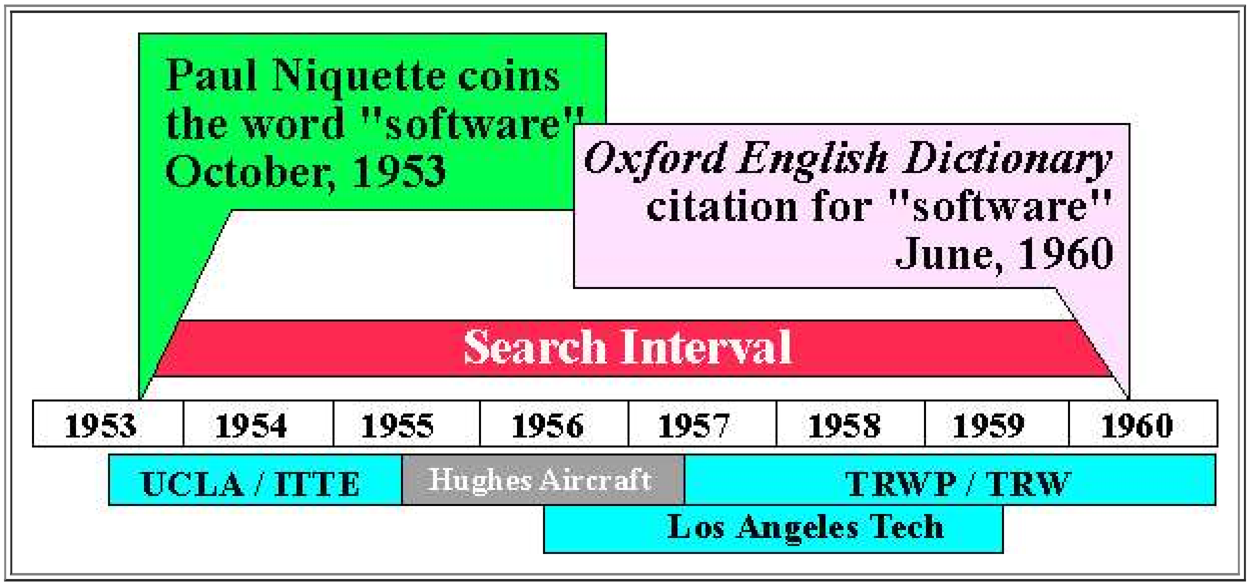
1. Published Documentation from the fifties in which I used the word 'software'.
2. Unpublished Documentation from that time in which I used the word 'software'.
3. Established Stories about the invention of the word 'software' in published sources.

1.

Part 5

Search Interval: The OED Guidelines

Guidelines for authenticating my claim were given to me by the Chief Editor of the *Oxford English Dictionary* on July 6, 2000, calling for three searches: Published Documentation, Unpublished Documentation, and Established Stories. As indicated in the diagram, to authenticate my claim, all three searches would be confined to the **Search Interval**, between October, 1953 and June, 1960.


Paul Niquette coins the word "software" October, 1953
Oxford English Dictionary citation for "software" June, 1960
Search Interval
1953
1954
1955
1956
1957
1958
1959
1960
UCLA / ITTE
Hughes Aircraft
TRWP / TRW
Los Angeles Tech

The most likely confirmations would be derived from my work at UCLA's Institute of Transportation and Traffic Engineering and from technical projects at Thompson-Ramo-Wooldridge Products (later TRW, Inc.). Then too, prospective sources included my students in analog and digital computers at Los Angeles Technical College.

1. Published Documentation

With limited time to devote to the softword project, I drew extensively upon the resources of the Internet. On-line historical databases, of course, favor events that occurred long after the 1950s. I contacted likely sources, including the Institute for Electrical and Electronic Engineers (IEEE) and The Software History Center, with disappointing results.

"Three moves is the same as a fire," the saying goes. My personal archives have been eroded by seventeen residence relocations plus as many career changes. A mere fifteen book-boxes remain. I arranged to have them shipped from storage in Macomb, Illinois to my home in Concord, California. I spread out their contents on my work-bench. One item, the draft of a long-forgotten paper that I submitted to the ACM caught my attention. I dashed into the house and gazed at my computer screen...

Sure enough, *OED*'s earliest citation was a quote taken from *Communications, Association of Computing Machinery* (ACM). I took that as an omen. In about 1959, I had submitted a "transaction" for one of ACM's journals. My short paper described an elegant random number generator for use by a feature-challenged computer with a drum memory (that's right, a drum -- please do not laugh). In particular, it was the RW-300, and I was using that meager machine to perform a Monte Carlo analysis of a proposed landing system for Chicago's Midway (not O'Hare -- see "Lottery Aloft"). I clicked up the ACM website, where I read...

> Founded in 1947, ACM is a major force in advancing the skills of information technology professionals and students worldwide. Today, our 75,000 members and the public turn to ACM for the industry's leading Portal to Computing Literature, authoritative publications and pioneering conferences, providing leadership for the 21st century.

...and requested a search of their archives for 1959. Bernard Rous, Deputy Director of Publications ransacked the records but was unable to find my paper. I got to thinking: Whereas the research work was completed in mid-1959, the paper would have been submitted no earlier than 1960, thereby coinciding with the citation already published in *OED*. I do remember being a mighty proud 26-year-old upon seeing my name in print alongside an algorithm for a computer model that profoundly influenced a safety-critical decision in aviation history; however, it is doubtful indeed that I actually used the word 'software' in that paper. Instead, I would have referred to it by the formal and most common terminology, "subroutine."

Among various newspaper clippings in my files dating back to the early fifties were reports of local service-club speeches, an article I wrote about UCLA's pioneering automobile crash-injury research published in *California Engineer*, a piece in *Westways* magazine reporting on the first radar speed-meter, and several formal papers. The papers were all related to traffic engineering and explicitly authored by the principal researchers for whom I worked. None mentioned computers, let alone 'software'.

The two most relevant papers among my keepsakes were published long after 1953. My heart skipped a beat when I found them. The years coincided with those in the *OED* entry for the word 'software'. A coincidence -- or another omen?

- "Design Considerations for Computer Driven Relay Controlled Displays," R. P. Niquette, American Institute of Electrical Engineers, June 20, 1960, 10 pp.
- "Terminal Area Sequence Control," R. P. Niquette, et al., a 230-page research report for the Federal Aviation Agency, February, 1961.

Alas, neither paper included the word 'software' -- not even *once*. It was like antibodies had formed around the word.

Worth mentioning, I think, is the fact that I did deliver earlier versions of both papers at various conferences, including one at University of Pennsylvania and another at Massachusetts Institute of Technology, from within the **Search Interval**. From the podium and on technical panels, I distinctly remember how I allowed myself to be less formal, even jocular, yet printed in a formal paper 'software' would have looked -- well, too *un*serious.

So much for authenticating my claim by Published Documentation. Just the same, any surviving attendees who recall those sessions during which they might have heard the word 'software' for the first time have been invited to comment here. After six years, none have done so.

2. Unpublished Documentation

Back to my workbench and those fifteen boxes.

In 1953, I was a junior at UCLA. Accordingly, the Unpublished Documentation in my archives from that period are nothing but notes and term papers on engineering subjects. There were *no* courses in computers at UCLA until after I graduated in 1955.

My first necktie job put me on secret projects for Hughes Aircraft, and, apart from private maunderings and satirical cartoons, there are few unpublished documents in my unshredded personal files. One cartoon I was hoping in vain to find would have characterized the time for readers of today.

As for lecture notes for my computer class at Los Angeles Tech -- well, there never were any. That's right, to this day, I have always taken pride in speaking from memory, extemporizing problems, stimulating discovery.

Now, it has been a lifelong habit to mark ideas as they occur to me with a stylized lightbulb in the margin of my notebooks. A few loose-leaf entries from the fifties include sketches so marked, which in their time were doubtless viewed as brilliant. Over the years and in the face of technological advances, however, the embarrassing obsolescence of those early efforts all but assured that they would be discarded on residential moving days.

A few escaped, and naturally, I was hoping to find among the surviving pages a lightbulb alongside the word 'software'.   That would do it for *OED*, I told myself -- Unpublished Documentation from the fifties. But no. Whereas in my undiscarded papers, the word 'software' appears routinely in many places, usually abreviated 'S/W', the earliest occupies a line on a page dated 11/17/63 when I was at Scientific Data Systems -- beyond the **Search Interval**. No lightbulb for that one.

The third and last search was about to commence...

3. Established Stories

There ought to be many of these. All I need to do is find a few of the story *tellers*. That's what I thought, anyway. Three groups of prospective narrators are most appropriate for this last search.

3a. Family Members 1953 onwards
3b. Colleagues at UCLA 1953-1955
3c. Students at Los Angeles Tech 1956-1959

3a. Family Members...

...may be the least credible informants under the harsh scrutiny of *OED*'s editorial staff; however, the omission of their retrospective comments might be suspicious, too.

As a working full-time student with a long commute to the UCLA campus, I devoted precious few waking hours to my family. A half-century later, there are no letters-in-the-attic to discover, either -- only a handful of greeting cards among my keepsakes, hardly the place to find Established Stories about the word 'software'.

My mother died in 1975. In 1953, she was a single mother of three juveniles with little time to be curious about her eldest son's academic exploits. My father died in 2002 and would have been a likely source for Established Stories about me and the word 'software'; however, he was estranged from the family for ten years beginning in 1953.

> During a telephone conversation in 2001, my dad allowed that the word 'software' probably did first reach his ears from me during a television interview I gave in New York (in 1958). He offered to sign an affidavit to that effect. I procrastinated, thinking -- wrongly -- that there would be plenty of time for that if the need arose.

In an effort to reprise Established Stories about the word 'software', I wrote a delicately worded e-mail message to my sister and two brothers:

> July 13, 2000
>
> Dear (in alphabetical order) Alan, David, and Patricia,
>
> Here is a long shot. I am working on personal project with which each of you may be able to help (large values of "may" and small values of "help").
>
> Hoping that you will not be inconvenienced by my query, permit me to attach excerpts from correspondence with John Simpson, who is Chief Editor of the *Oxford English Dictionary.*
>
> Looking forward to receiving your comments, I am merely, your brother
> Paul

My sister, Patricia, now matriarch of our family, sent me a reply that brought tears to my eyes but no Established Stories to my search...

> July 24, 2000
>
> Dear Paul,

In 1953, I was 11.

I have few recollections of my life in the fifties except for flashes such as 1) learning how to roller skate down hill and negotiating a perfect circle where the sidewalk of Perkins Lane turned to Beland Boulevard; 2) learning to ride a bicycle; 3) listening to "Weemaway" on the car radio on the way to the beach; 4) learning where middle C was on the piano keyboard and where it was on the music; 5) learning to play four-hand chopsticks on the piano; 6) watching college guys skindiving to retrieve abalone; 7) enrolling in private school.

You were the sponsor of all this and much more. I wish I could be of some help on this laudible cause.

Love,
Sis sib

My two brothers, David and Alan, were respectively 9 and 10 years old in 1953. Today, both are distinguished members of the clergy, David in Colorado and Alan in Oregon. I was eager to have their replies, hoping they would include helpful contemporaneous recollections. The closest to Established Stories, albeit from members of my family, may well be inferred from the following letters from my brothers:

July 19, 2000

Dear Paul,

In my recollection, you were the first one I heard use the word “software.” You took the time to explain to me the concept of standardized sub-routines, or utilities, as you gave me an after-hours tour of your workplace in the summer of 1957.

I have recounted the details to many of my engineer and technical writing friends at Hewlett-Packard as a matter of interesting conversation to prove I had some sense of the earliest days of the computer revolution. (I’m admitting to grasping after a little reflected glory from an older brother -- pretty humble, huh?)

Here are some details of the context of the event. I remember the day you took Alan and me to the Ramo

Wooldridge Building south of LAX. It was after hours, but with your security card we gained entrance and took a tour of the facilities. You led us into the climate controlled room which contained the RW-300 computer – roughly the size of a large chest-type freezer. Today, I understand that many $30 hand held calculators at Walmart can perform the same computations more quickly. How times and technology has have changed!

While you were explaining the technology of this mammoth calculator, you showed me the perforated paper tape that I now understand to be a primitive programming means. You distinguished this input tape from the machine itself, referring to the latter as “hardware,” and the former as “software.” This was the first time I had heard this term and the concept behind it.

You explained that the perforated tape was designed to give instructions to the computer regarding certain basic utilities or sub-routines that a designing engineer. (I’m not sure if they called them programmers back them.) This designer could invent instructions to set up the computer to function as he desired. The advantage of the paper software, you demonstrated, was its transportability to similar “hardware” units, so that the value of an expensive program could be shared and expanded without the need for replicating the design from scratch for each computer.

I have also enjoyed recalling to my technical friends that your office was next to that of Simon Ramo. Ramo and Dean Wooldridge later merged somehow with Thompson Products, you told us. While Thompson Ramo Wooldridge does not ring a bell with many, its acronym does – TRW!

Paul, you certainly shared from the earliest days the adventure in an industry which has so revolutionized the world.

Warm regards,
David

September 14, 2000

Dear Paul,

To the best of my recollection, it was back in the early 1950's, when our big brother Paul would take us to the beach and to see various sights, like the "Pigeon Hole Garage" in downtown LA. I remember one time when you used the word "Software." We thought you were making a joke. (You were always able to make us laugh, even during some hard times.) Even at a very young age, we were familiar with the idea of a "hardware" store, since we were always working on one project or another. You made a noble attempt to explain the difference between "hardware" and "software." But it was not until many years had passed that I came to understand and appreciate the significance of that word.

With the proliferation of affordable PC's in the 1980's and 90's, I finally got tuned into the difference between the computer hardware I purchased, and the software -- the intangible operating systems -- that made various programs work. What you tried to explain to us in the 50's finally made sense. Since the early 1980's, I have often told friends and acquaintances, "My brother Paul coined the word 'software'."

Back in the mid-1990's, when my own Kyndi Joy was a tiny little tyke (a first-grader), as I was driving her to school one morning, she asked ponderously, "Daddy, what is 'silverware'?" I told her that silverware included things like the knives, forks and spoons we use at the dinner table. Then I queried, "Why do you ask?" Kyndi replied, "I keep hearing you tell people that your brother, Paul, invented silverware." I explained to her that the word was "software," and she now understands that her Uncle Paul is not the sole inventor of software, but the man who coined the word.

Paul, I will always be proud that my older brother has contributed a word to the English language that is now spoken around the globe.

Yo Bro,
Alan

The phrase "summer of 1957" above is possibly sufficient for the editors of *OED* to establish my claim at least for *usage* prior to 1958 -- a date that has actually become more significant than the 1953 date for the coinage, as will be seen. Adding gravitas to that supposition are a few encouraging comments I have received from long-time friends and working associates, especially this one from Al Bongarzone:

> August 21, 2000
>
> Paul,
>
> I started as a programmer working for the RAND Corp at MIT's Lincoln Lab in 1956. The term "software" was in general use at the time. I didn't realize that among your many other accomplishments you had also coined that term, but I am not surprised, as your facility with words is widely known and admired.
>
> Best regards in this latest challenge.
> Al

3b. Colleagues at UCLA...

...fellow students and university staff -- they were all right there in 1953, present and in their adulthood for the historic moment of coinage and first usages of the word 'software'.

Over the years, I have maintained contact with only one fellow engineering student, Don Lauria. He reminded me that he had no interest in "giant brains" during the early fifties. Decades later, though, Don became a computer entrepreneur, selling and applying software-intensive data processing systems for small businesses. With regret, Don admits to having no recollection today of when or where he first encountered the word 'software'.

With some effort, I tracked down another fellow student, Angelo DeGrace. In view of my neglect for our friendship, I was not comfortable asking him up front to ransack his recollections about the word 'software'. Later on, when I was leaving on vacation, I called him for a chat. He was just entering his third session of chemotherapy. When I returned, Angelo DeGrace was gone.

Ten persons on the staff at UCLA in 1953 would be the most likely narrators of Established Stories...

Paul Barber	Heinz Haber
Al Berg	Slade Hulbert
Lewellyn M. K. Boelter	Dave McClinton
Bob Brenner	Derwyn Severy
Dan Gerlough	DeForest Troutman

- Eight have died, including my mentor Dan Gerlough and my chess partner Heinz Haber. Both would be centenarians by now. The discovery of each obituary struck me with no small amount of sadness, as I have been forced to cope abruptly with the collective loss of the finest, most generous persons I have ever known.
- Dave McClinton, who was never a close friend, is retired in Northern California. He responded to my e-mail queries with grumpy non-sequiturs, possibly embarrassed to be asked to test his memory.
- Slade Hulbert, one of the founders of Human Factors Engineering, is now retired in nearby Danville, California and a confirmed Luddite. I eagerly arranged a reunion. Over wine and cheese, our conversation was dominated by his geriatric ailments. I commenced to reminisce with Slade about the software I wrote for his projects, but he interrupted me. "I once tried to program the SWAC myself, Paul. I decided I hate computers." I left a self-addressed stamped envelope for Slade's convenience. That was more than five years ago.

3c. Students at Los Angeles Tech...

...would have heard their bespectacled instructor use the word 'software' more than once, making them all likely sources for Established Stories, one might expect. There were more than a hundred of them, and I may have kept those rosters for a dozen residential moves.

Foremost, there were four students in one class who were all from TRWP (now TRW). They pulled their neckties loose and sat together in the front row taking copious notes...

1. It was easy for me to tell that Ray Jacobsen was the boss of that group ("I'll go bring the car around, Mr. Jacobsen," said one after class each evening). Ray was a division vice president at TRWP and hired me during that semester ("assuring myself of a passing grade," he would joke for years thereafter). He went on later to found Anderson-Jacobsen, the leading modem manufacturer for its time. Ray would be well into his eighties by now, which may explain why I have not been able to locate him.
2. The next student in the hierarchy was Lou Perillo, sales manager at TRWP. Enjoying a distinguished career, he continued as a colleague and friend through most of my years at Xerox. Lou died back in the seventies.
3. Then there is Rigdon Currie, who became a close friend. He has continued to work with me on various business deals off and on for four decades. A birdwatcher of some renown, Rig is now retired in Point Reyes and pleads senility whenever I mention the word 'software'.
4. Finally, there is the exceptionally talented storyteller, Ray Stanish -- my best hope for a first-hand collection of Established Stories! Please note the exclamation point. After all, Ray made a lucrative career out of comic impressions of Einstein and popularizing computers in hilarious speeches all over the country, retiring at the age of 32.

All right, damn it, so Published Documentation does *not* appear in journals, and Unpublished Documentation has *not* survived in my archives; likewise, Established Stories in the 1953 time-frame, whether from family or UCLA associates, are *not* forthcoming from living memories. I am *not* giving up.

An "adventure" that started out as a casual investigation ostensibly on behalf of my progeny had been fraught with set-backs and disappointments and now was becoming an ego-driven obsession. Forget October, 1953. Anything before 1960 -- *OED*'s earliest citation -- will do. I took a deep breath and decided to launch an all out search-and-query mission targeting my other colleagues at TRW. I ransacked my memory and files and came up with thirty names from 1957...

Bill Aiken	Fred Holland	Monty Phister
Elliot Beiderman	Curt Johnson	Elliot Rech
Emil Borgers	Harry Keat	Ed Robin
Elaine Brooks	Wayne King	Stu Schy
John Dauwalder	Paul Likins	Francis Semere
Jim Farzan	Joe Manildi	Jack Smeltzer
Fred Fielding	Dan McGurk	Tom Stout
Peter Friedlander	Bert Newman	David Tanz
George Heilborn	Harold Ottosen	Jim Trapp
Herb Henderson	Bill Paine	Lew Ward
Dick Hexter	Steve Pardee	Bob Wilmer

Tracking them down one by one in a crescendo of correspondence and follow-ups, I encountered more obituaries and more memory lapses. Still, there was no turning back from my efforts to harvest pre-1960 Established Stories. I continued writing and calling and asking.

Suddenly the endeavor lost its meaning!

Part 6

Challenger: John WilderTukey

See April 2013 Update:
Discovery of a Paper Published in 1956 Containing the Word *Software*

Subject: Provenance for the Word 'Software'
Date: Sat, 29 Jul 2000 07:36:15 -0700
From: Paul Niquette <paul@niquette.com>
To: NYT Editors <letters@nytimes.com>

Dear New York Times Editors,

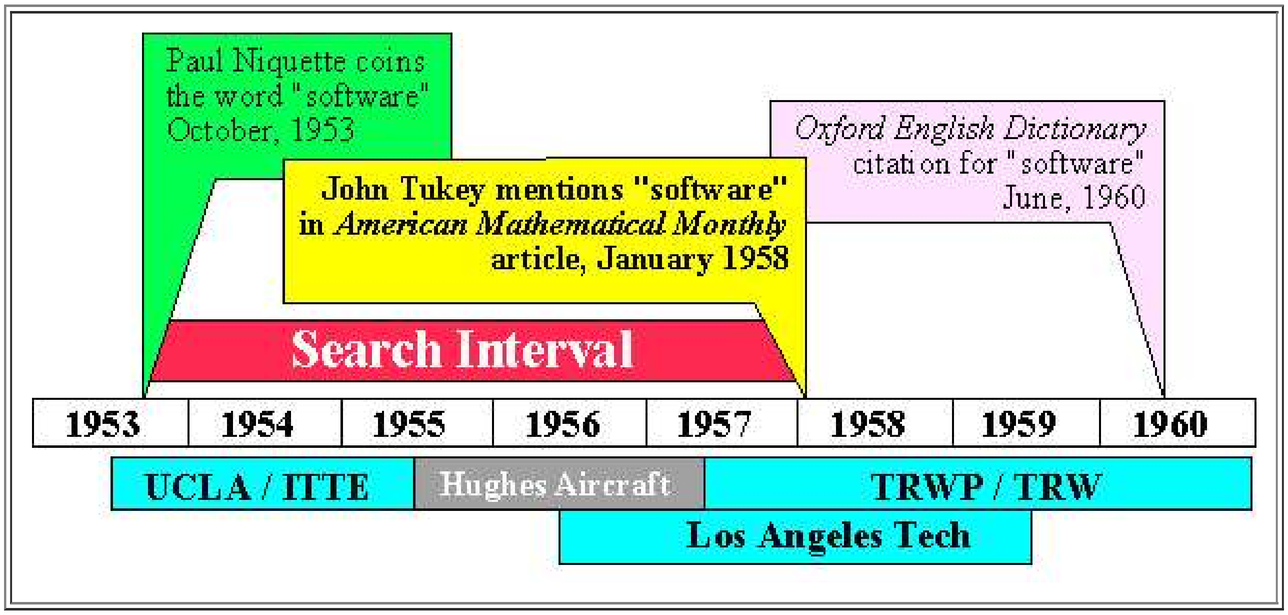
Quoting from yesterday's obituary: "John Wilder Tukey, one of the most influential statisticians of the last 50 years and a wide-ranging thinker credited with inventing the word 'software' [citing a 1958 article in *American Mathematical Monthly*], died on Wednesday in New Brunswick, N.J. He was 85."

Be advised, I coined the word 'software' in October 1953. For the past four months, I have been gathering documentation and reports of witnesses to support my claim.

If further informaton is required, please do not hesitate to contact me.

Best regards,
Paul Niquette

The immediate effect of the Tukey obituary upon my "adventure" was to cut off the Search Interval for authenticating my claim, at January, 1958.



Paul Niquette coins the word "software" October, 1953
John Tukey mentions "software" in *American Mathematical Monthly* article, January 1958
Oxford English Dictionary citation for "software" June, 1960
Search Interval
1953
1954
1955
1956
1957
1958
1959
1960
UCLA / ITTE
Hughes Aircraft
TRWP / TRW
Los Angeles Tech

But of course, I began thinking about all those inquiries I had launched, hoping to contact surviving TRW colleagues, all those phone calls to old friends, all those follow-up letters and messages. Get real, I told myself. Unless datable no later than the end of 1957, any Established Stories thus obtained would be worthless.

Same for any of my LA Tech students from the fifties who might stumble onto my website and remember... Remember what? To contribute a relevant anecdote, an LA Tech student...

- must remember back 44 years,
- must remember attending my class in one of exactly three semesters,
- must remember hearing me say the word 'software' during at least one lecture, and
- must be certain that the incident was the first time he or she had heard the word 'software' -- this despite intervening decades of hearing and reading, saying and writing the word 'software'.

It does not take a statistician of the calibre of John Tukey to assess the joint occurence of such unlikelihoods.

One student and close friend, Rig Currie, did act on my behalf by writing an e-mail to another student, Ray Stanish: "I hope you remember the precise date of his use of the word 'software.' Would you please just lie if it's not perfectly clear. It would make Paul a happy man to have one reliable witness. The rest of us have brains too fried by alcohol and dirty thoughts to have room left to even lie effectively." The appeal was unavailing.

So much for Established Stories, the third and last search. October of 2000 marked the end of the softword project. There would be no entry in *OED* or any other dictionary giving me credit for coining the word 'software'.

It was for such a moment that the s-word was invented. The four-letter one.

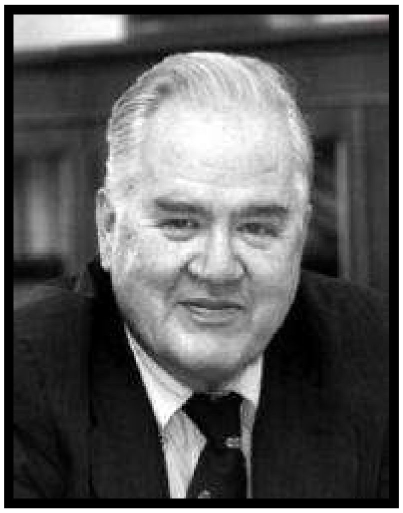
Three years went by. A few replies trickled in from friends and family. The **Softword** page on my website just sat there endlessly proclaiming itself to be a work-in-process, occasionally drawing correspondence from near and far. Most notably was Fred Shapiro, the library scientist who achieved worldwide acclaim for discovering the 1958 reference cited in the Tukey obituary. He wrote from time to time hoping to sell me his free-lance services perusing magazine morgues.

There can be little doubt that the years 2001 through 2003 will be characterized by future historians as global downers. Still, in that hiatus I found many cheerful distractions. I published the Internet versions of two books, *A Certain Bicyclist* and *Squawk 1200*, added a dozen entries in *Puzzles with a Purpose*, and launched yet another new career -- number eighteen since 1953, by my count. My life trajectory has been nothing but discontinuities, it seems.

October 2003 came along, and I marked the 50th anniversary of the epiphany with a rueful chuckle. The softword project had been an adventure, all right -- more than an adventure, in fact. The search for Established Stories connected -- *re*-connected -- me to projects and people of great personal value. Never mind that the projects have been obsoleted and that so many of the people have departed. All have returned with full vitality to my mind exactly the way they were decades ago. If that is the only prize I am justified in seeking, then...

> Wait just a damn minute. Forget sentimentality. I have a challenger!
> Exclamation point intended.

The Challenger



There is no denying his eminence. Biographical sketches and tributes abound on the Internet. John Wilder Tukey's formal work fills volumes, shaping statistical analysis. He has served in prestigious posts at all the top venues, including Princeton, Bell Telephone Labs, and performed a number of major consulting assignments, including Educational Testing Service, Merck & Co., and, ironically, Xerox, where I faintly recall meeting the man in the '70's.

Tukey's popular fame was assured by such high-profile work as projecting the election-day results of presidential contests for national television, plus...

- In the 1950s, he publicly criticized Alfred C. Kinsey's research on sexual behavior as being seriously flawed because it relied on a sample of people who knew each other. Professor Tukey said a random selection of three people would have been a more representative than a group of 300 chosen by Kinsey.
- In the 1970s, Professor Tukey was chairman of the research committee that warned of the damage to the earth's ozone layer caused by aerosol spray cans.
- In the 1990 Census, he recommended the use of statistical formulas in order to count poor urban residents whom he believed had been missed.

Most relevant to my little softword project, was the way Tukey's use of the word 'software' is headlined in the New York Times obituary, which described John Wilder Tukey as...

> ...an amateur linguist who made significant contributions to the language of modern times. In a 1958 article in *American Mathematical Monthly,* he became the first person to define the programs on which electronic calculators ran...Three decades before the founding of Microsoft, Mr. Tukey saw that "software," as he called it, was gaining prominence. "Today," he wrote at the time, it is "at least as important as the 'hardware' of tubes, transistors, wires, tapes and the like."

It's as if one of Tukey's achievements was expressed in what appears to be a mere off-hand mentioning of the word 'software.' That observation struck me hard and set me to wondering...

> **Why did John Wilder Tukey go to his grave without actually claiming the invention of the word 'software' 42 years earlier?**

John Tukey (1915-2000) was a contemporay of my father, Reaman Niquette (1913-2002). Over a period of three years following his death, John Tukey has become an admired friend of mine. That is not meant facetiously. I am embarrassed to admit this, but while studying his life and his work, I have heard myself in simulated conversations with Tukey.

> "Excuse me, John, but I just don't think you got it!"
>
> He looks at me from his picture there and shrugs, "You want credit for the word 'software'? Be my guest."

The editors of *OED* will doubtless accept that conversation as evidence of my eccentricities more than for authentication of my claim.

There is no evidence that Tukey ever declined the acclaim to which he was entitled. Nevertheless, from all I have seen, for decades after 1958, Tukey manifested total indifference to the word 'software,' arguably **the most important new word of the 20th Century!**

Editors of any dictionary, especially the *OED*, would have eagerly glommed on to documentary evidence for the origin of such a far-reaching entry and would have been thrilled to publish that 1958 citation from so eminent and colorful a figure of the Twentieth Century. Moreover, it goes without saying that for his formal writing, Tukey would never have chosen to have his work published anonymously. The *OED* protocol, however, now casts Tukey in the role of a posthumous claimant. That makes him a challenger to my claim. Thus, after three years, I became tempted beyond my powers to resist. I set about to demystify John Wilder Tukey's reluctance.

Let us begin with his non-software coinages. Tukey has been credited with several. According to one admirer, Manfred Schroeder, Universitaetsprofessor Physik Goettingen, Germany...

> Tukey was also a great wordsmith: he coined the terms bit, byte, software and cepstrum, (the Fourier transform of the logarithm of the Fourier transform). But some of his kookier coinages, like quefrency (for cepstral frequency) and saphe (for cepstral phase) didn't catch on.

The word 'bit'? -- possibly, although Claude Shannon was first to publish, as indicated in the *OED* entry...

> **1948** C. E. Shannon in *Bell Syst. Techn. Jrnl.* July 380 The choice of a logarithmic base corresponds to the choice of a unit for measuring information. If the base 2 is used the resulting units may be called binary digits, or more briefly *bits*, a word suggested by J. W. Tukey

...but probably not 'byte' according to the *OED* entry**...**

> **1964** Blaauw & Brooks in *IBM Systems Jrnl.* III. 122 An 8-bit unit of information is fundamental to most of the formats [of the System/360].

By the way, as the word 'bit' caught on, some computer scientists became indignant. They saw Professor Tukey as an outsider. "Not everyone was happy that he was naming things in their field," said Steven M. Schultz, a spokesman for Princeton. That complaint is not justified, in my opinion. Computer scientists have been singularly *un*creative in naming things, beginning with 'computer science' itself. And 'bit' is as well suited to communications as to computers anyway. Meanwhile, Tukey's neologisms within his own field ('cepstrum,' 'quefrency' and 'saphe') do not appear in the *OED*, but then neither do some of mine.

Turning to the now famous sentence, which appeared near the beginning of Tukey's essay titled "The Teaching of Concrete Mathematics" in the January 1958 *American Mathematical Monthly*...

> Today the "software" comprising the carefully planned interpretive routines, compilers, and other aspects of automative programming are at least as important to the modern electronic calculator as its "hardware" of tubes, transistors, wires, tapes and the like.

Given the formidible practicality of software to *do* things not just help programmers write programs to do things; given the manifold impacts of software upon all aspects of human work and achievements, of life and now even of play; given the astounding realities of software, which among all of mankind's creations is uniquely indestructible -- given all those amazements, where is the *grandeur* in Tukey's 1958 statement? There isn't any, only a routine declarative-in-passing.

Consider the word 'appliance.' Suppose the earliest published use of that term read as follows:

> Today the "appliance" (of electricity) comprising the carefully crafted numerical control machines, automatic

milling machines, progressive-die punch-presses, industrial robots, conveyor belts, and other implements are at least as important to the modern toolmaker's trade as the "workshop" of bench, floor, walls, files, drill-bits and the like.

Um...what about radios and reading lamps, vacuum cleaners and ventilating fans, washing machines and water pumps, central heating and air conditioning, garage door openers and clothes dryers, elevators and escalators, televisions and videos? *Things*, not just the means for making things.

> Readers may have noticed my deliberate practice in this work of using single quotes to set off the word 'software' from software itself. The reader is invited to notice Tukey's use of the double quote around "software" in a sentence in which software -- not the word 'software' -- was being described.
>
> By convention, double quotes indicate that somebody is being quoted. Oh sure, double quotes can also be used to denote sarcasm (The "non-toxic" effluvia from that plant is killing my begonias), but sarcasm seems unlikely for the case at hand. Accordingly, those double quotes do *not* make sense in that sentence -- unless somebody is being quoted, which, if I prevail with my claim, is a distinct possibility.
>
> It is an unmistakable fact that the double-quoted "software" was being taken for granted by Tukey, much as the double-quoted "hardware" in the same sentence. Both receive suffixed explanatory passages, suggesting that in Tukey's mind, a 1958 reader would not have otherwise known what "software" means -- but then, what about the double-quoted "hardware"?

If indeed this "momentous" sentence (sarcasm intended) is to be credited with bringing the word 'software' into the English language, then one might be forgiven for expecting a prefixed phrase of the form, "*What I call* 'software'..."

Is it not clear by now that the protocol used by *OED* for accreditation is anomalous? Publication of a word is not the same as coining a word. There is no denying Tukey deserves credit for the former, and he may indeed deserve credit for the latter, but where is his *claim*? That off-hand sentence in the 1958 citation could well have been written years after the first use of the word 'software', which is my assertion exactly.

The way I see it, after decades of historical developments facilitated by software -- the worldwide emergence of what I call '**The Software Age**' -- John Wilder Tukey ignored the immense implications of the word 'software' decade after decade and went to his grave knowing that he was merely using an established word.

So then, who really *coined* the word 'software'?

.

Part 7

Question: Fair is Fair

Fair is fair. The question posed regarding my challenger...

> Why did John Wilder Tukey go to his grave without actually claiming the invention of the word 'software' 42 years earlier?

...with some modification, applies to me. Considering what is known today -- indeed, given what the world has known since the sixties about software, about its manifold impacts on every technology, about its power to shape modern life...

Why would the person who coined the word 'software' wait a half century before claiming it?

The previous sentence deserves an exclamation point more than a question mark. I shall refer to it from this point onward as **the why question**. Anyone called upon to judge the assertion I am making here is entitled to the best retrospective answer to **the why question** I can give. Answers plural. Permit me to frame them in decades.

1950s

The most critical interval for documenting my claim, of course, will necessarily be the 1950s. Alas, those youthful years blasted by in a blur of activities -- years filled with distractions and impediments both personal and professional, affording me abundantly plausible answers to **the why question**. Throughout that decade, I simply had no thought of 'software' as being anything to claim. Simple as that.

As described elsewhere, the hardware used in my high school projects with symbolic logic was limited to relay technology and programmed by patch panel. It was a discrete-value analog computer. Until 1953, like most people at that time, I had never seen a stored-program digital computer -- a von Neumann Machine. Then, while in my junior year at UCLA, I was actually paid to program the SWAC.

It was in October of that year, that I coined the word 'software' more or less as a prank. I was 19 years old. I never expected the term to be taken seriously. For the next two years, I used 'software' in dozens of speeches and lectures, accompanied necessarily by its definition. Shrugs and smirks hardly provided an incentive to accept -- let alone seek -- any kind of credit for the word 'software'. There, for the record, is my earliest answer to **the why question** -- my own dismissive state-of-mind, not to mention frequently frivolous post-adolescent behavior (see sniglets). Which is not to say, that I did not perceive -- even at those earliest moments -- the wonderous, world-changing power of software. It's just that I thought the word 'program' was serviceable enough for serious speaking and writing.

The main body of my work at UCLA was in automobile crash injury research and radar speed meters, both a far cry from anything relevant to **the why question**. All publications that referred to the computer programs I wrote, of course, received attribution to the principal researchers (Dan Gerlough, Slade Hulbert, Harry Matthewson, Bob Brenner), who were warily disinclined to incorporate 'software' in any erudite writings that had been drafted by a mere undergraduate. Then too, from the very beginning, I regarded the word 'software' as a misnomer. Even at the age of 19, I sensed the unique *hardness* of software (see Software Does Not Fail). There is nothing created by mankind that is more enduring!

For the first two years after graduating from UCLA, I worked at Hughes Aircraft on *secret* projects, including radar sets, air-borne fire control systems, and guided missiles. All applied continuous-value analog computers. Still in my early twenties, I was not allowed to produce technical papers on computer hardware. Few people indeed were then writing about software, which would have been called programming anyway. That answers **the why question** for those years in terms of contemporaneous publications. I kept up my private studies on "finite automata" and their applications, though, and accumulated something of a local reputation. In 1956, I was contacted by State of California and received a contract to write the syllabus for the very first junior college course on computers, both analog and digital. The Department of Education awarded me an expedited credential, and I began teaching the course to SRO classrooms at LA Tech. Naturally, I spoke the word 'software' in my lectures but only informally, always as a term of distinction from 'hardware', giving the word 'program' the merits of invincibility.

My luncheon talks to Kiwanis or Lyons about "giant brains" were quoted occasionally in the local press, but I was careful to credit others with the wonderous inventions themselves and would never have sought recognition for the handful of words being appropriated to describe them ("you heard it here first, folks"), which is the most rational answer I can give to **the why question** for those occasions.

In 1957, Ray Jacobsen, then a division vice president at Thompson-Ramo-Wooldridge Products (TRWP, pronounced "twerp" -- happily renamed TRW Computers) was a student in my computer class, along with his marketing team comprising Lou Perillo, Ray Stanish, and Rigdon Currie. After a dozen hours of observing my antics, Jacobsen hired me away from Hughes to teach programming and maintenance courses on the RW-300, a digital computer designed by space scientists for application in the earliest real-time process control systems. These were the most serious of industrial environments, by the way. At TRWP, we were confidently programming primitive computers to run chemical factories, oil refineries, cement plants, and -- shudder -- nuclear reactors. Still in my mid-twenties, however, I continued to build a singularly unserious reputation. (see Debugging and Degaussing).

Eventually bored with the silly word 'software,' I used it sparingly myself, treating it more or less as slang, but then in the late fifties I began *hearing* it occasionally -- at first only from my students, later from customers and suppliers. I remember well that I found it amusing if not ironic. The word 'programming' continued in use, enjoying whatever respect the profession could garner for itself, while the word 'software' was often pejorated: occasionally 'software' was even treated as a verb ("we'll just have to software around that problem").

At the Eastern Joint Computer Conference in New York in 1958, I was alone in the TRW booth when a television news crew showed up. I was standing alongside the company exhibit, the only one with a fully functioning computer system on display. I gave an interview about the RW-300 and its innovative applications. By coincidence, my father, whom I had not seen in a half-dozen years, stood grinning in the crowd, face iridescent. After pointing out features of the hardware, I unabashedly described the company's "real-time programs as what we call 'software'." Nearby shaking his head and wincing stood Lew Ward, head of public relations at TRW. I felt the smile disappear from my face. Thus, the answer to **the why question**, both then and later, included an element of jocularity on my part. One interview I remember most vividly was for "Voice of America." Afterwards, as an act of bravado, I tormented the public relations staff by reporting that I had made the remark, "Only in the most backward countries are they replacing computers with people." I then hastened to reassure my wide-eyed colleagues that the on-air translation of my words into Serbo-Croatian would not have included the word 'software'. Whew.

In 1958 while still at TRW, I became a consultant to the newly founded Federal Aviation Administration (FAA) at the National Aviation Facility Experimental Center (NAFEC) in Atlantic City, New Jersey, doing fundamental research in air traffic control systems. There I wrote programs for the RW-300 and IBM 704. I gave four software-intensive papers on man-machine simulators, track-while-scan radars, and computer-driven displays. My work was published in the proceedings of the AIEE. The NAFEC research is covered in a separate memoir (see *Squawk 1200*).

With respect to those solemn writings, of course, the answer to **the why question** is retrospectively obvious to any fair-minded reader: claiming coinage for the word 'software' was simply not appropriate to the subject matter and would have achieved nothing beneficial for my professional aspirations. Nevertheless, I vividly recall at least one formal occasion during which I spoke the word 'software' in a conference at Pennsylvania State University. Even at the podium, I did not think to attribute the word to myself (see Part 3).

1960s

For the early sixties, there is this one answer to **the why question** I might really like to offer -- that it would have been extremely immodest for me to claim the invention of a word with such rapidly widening currency. In all candor, however, my self-restraint came from an expectation that 'software' would soon fade from the lexicon of the industry, replaced, most likely, by "computer science."

My engineering work at that time comprised both hardware and software, including custom applications of the CDC-160A manufactured by Control Data and on-line stock-price delivery systems, trade name "Quotron," for the financial industry produced by Scantlin Electronics. I would hear the word 'software' occasionally, but it was not yet a fully accepted word in the computer field. Surely that is enough of an answer to **the why question** for this interval. But there is more, much more.

Real engineers designed and built *hardware*. I firmly recall that 'software' was regarded by most people as a sissy word, and managers treated programmers as second-class citizens. They skulked about with their coding sheets and punched cards, each decidedly loathe to characterize his or her own endeavours with such a marginal term as -- ugh! -- 'software'.

In 1963, I joined Scientific Data Systems (SDS), one of the "Seven Dwarfs" -- manufacturers of computers, all scrambling for a share of the market owned 80% by "Big Blue" (IBM). At SDS, I got undeservedly promoted and took up the struggle against the stereotype of a good engineer becoming a bad manager.

Plunged into high-intensity, administrative assignments, I formed and managed several hardware engineering departments (peripheral controllers, manufacturing testers, memory systems, design automation); I was responsible for the development of computer-based industrial control products, systems for the man-in-space program, and custom equipment for computers. With the appearance of "monitors" and later "operating systems," software gradually emerged from its status as a necessary evil and became a sub-industry, with growing economic importance. As for **the why question**, during the early sixties, practitioners of software were widely scorned for their lack of discipline. After all, they persisted in scuffing about in sandals and unkempt long hair. Enough said.

IBM in 1964 introduced the immensely successful System/360 and in 1968 made the fateful decision to "unbundle" their support services -- read 'software' -- from the hardware pricing itself. Given IBM's dominance of the computer industry, these two events probably did more than anything else to establish the word 'software' in popular vocabularies.

The reader is here invited to observe that my predicament with respect to **the why question** and everything else in my life changed abruptly 1969. For a record-setting price of a billion bucks, SDS was acquired by the Xerox Corporation, hardly a paragon of software technology. With aspirations to attack IBM head-on, Xerox gave SDS the name Xerox Data Systems (XDS), which then became a forlorn passenger in steerage aboard the Great Ship Duplicator.

1970s

Almost immediately after the acquisition, the giant hand of Xerox reached down and plucked me out of XDS in California and plugged me into a new career at corporate headquarters in Stamford, Connecticut.

Picture in your mind a nasal-retentive, unsophisticated engineer replete with pocket-protector and horn-rim glasses, recently sobered by horrendous adaptations to line management responsibilities, now suddenly transported into the politically nuanced world of corporate staff work and put in charge of a think tank, surrounded by a council of Ivy League authorities -- a priestly class of business futurists in three-piece suits and wingtips, who spoke in hushed tones and bowed to each other. I was not qualified to tie their shoes, but -- hey, I was their leader.

Concealing bewilderment in brashness, revealing impatience with unsubtlety, I quickly established my insufferable reputation as the "Coyote from the California." Eight years later I would be given the "lateral arabesque" then fired -- but I'm getting ahead of my story.

Known throughout the company, without affection, as "The Kitchen Cabinet," the seven of us commandeered the corporate jet fleet and conducted business reviews all over the world. Our meetings applied "The Rational Process," and we gave authoritative speeches and lectures in every university and business school you can name on "Financial Planning and Control," "The Architecture of Information," and "The Office of the Future." My specialty was "The Post-Petroleum Age" and later "The Software Age."

In addition to our collective oversights (I preferred the expression "overview responsibilites") on behalf of the corporate management, The Kitchen Cabinet was held accountable for forecasting the business implications of emerging technologies. There were many of those in the siliconizing seventies -- plus software, of course, unquestionably the least understood topic at the time, in the deep-pile, hallowed hallways of Xerox and, along with microprocessor-based hardware, terribly unsettling to the status quo. Xerography had long been dominated by an establishment steeped in photo-receptors and doctor-blades, stepping switches and relay logic, where "high-tech" meant a photo-diode instead of a mechanical lever.

Oh right, and it was a high-stakes game. The enterprise had aspirations to grow new revenues at the staggering rate of one billion dollars per year per year! The repetition at the end of the previous sentence is intended and so is the exclamation point.

In retrospect, it is fair to say that The Kitchen Cabinet fulfilled its mandate. We delivered closely reasoned insights about the future and sound guidance for scaling mountains of emerging business opportunities, which were, alas, nervously made into molehills by near-term financial mentalities. Don't get me started.

Too late. Most relevant to the present subject were the visionary concepts my colleagues and I championed at the Palo Alto Research Center (known worldwide even today as PARC), which included several transcendent innovations: the mouse and the graphical user interface (GUI), Ethernet and peer-to-peer communication protocols. These Xerox inventions and others were leaked -- indeed given away, as the record shows.

> A parade of entrepreneurs visited PARC through the '70s. Among them were the two Steves -- Jobs and Wozniac -- who founded a computer hardware company incongruously named after a fruit, which went on to exploit these technologies in commercial products. Oh right, and another visitor came by for a tour of PARC, a soft-spoken, bespectacled chap named Gates, who went on to found a company that actually tried to sell software products separately from hardware.

No matter. PARC's magnificent developments were dismissed repeatedly by Xerox financial managers for not meeting bottom-line business criteria relevant to reprographics (see Thicket in the Bilge). Besides, there was all that talk about -- well, software. For a landmark review of these events see *Fumbling the Future: How Xerox invented then ignored the first personal computer* by Douglas K. Smith and Robert C. Alexander, iUniverse.com, Inc. 1999.

To the chagrin of both IBM and Xerox, the personal computer took center stage. Moreover, for Xerox, networking PCs together was only going to impact the market for reprographics, which was exploiting the worldwide paper deluge. With a corporate sigh, one might imagine, IBM appropriated Charlie Chaplin and in brilliant campaigns preserved their primacy in computers, this despite the impudent intrusions of Apple. Meanwhile, Xerox indulged in denial, a perilous policy in technology. My heavily colloquialized business advice was "If you ain't got it, you gotta get it, or you gonna get got by it!" Corporate budgeteers with xerographic toner under their fingernails rejoindered with, "Apart from smoke signals, no form of communications has ever been displaced by a successor."

If you can envision the likely reception at Xerox for any publicized claim by me for inventing what had become a politically incorrect word, then you have your answer to **the why question** for most of that decade.

1980s

Dropped out of the Xerox stratosphere, I parachuted into a raging battlefield of high-risk entrepreneurial adventures, including a disk-file manufacturing start-up (NIS), a diversified patent licensing service (APT), a gaming machine development company (G/N), a video production house specializing in animation and special effects (VCS). Don't bother to look them up. Suffice it to say, I paid my tuition for on-the-job training in capital formation.

With my personal resources depleted, I eagerly accepted a position as chief engineer at Computer Automation, where I took on the responsibility for leading 130 professionals in the "Naked Mini" division. Glad to have a salary, I was also especially pleased to be returning to my roots -- computer hardware. But everything had changed. Where were the "main-frames"? The machines I had known at XDS were now called by the retronym "maxi-computers," inasmuch as their successors were nicknamed "mini-computers." By the early 1980s, they were already being overtaken especially in the control market first by "single-board computers" then by "micro-processors" and finally by "microcomputers." Imagine that, a whole computer on a chip.

My short absence from fast-pace developments in computers put me in mind of a Rip Van Winkle, rubbing his eyes and shaking his head, bedazzled by "buttons and bottles." Hardware, though, was not my most difficult challenge. Before I had found the lavatory at Computer Automation, I was confronted by a decidedly transformed development world now dominated by -- ta-dah! -- 'software engineering'. That's right, I was suddenly surrounded by persons solemnly referring to themselves as 'software engineers'.

The origin of the word 'software' was far from my mind by then. Besides, what was I going to do? -- ask one of my software engineering managers, "Ever wonder where the word 'software' came from?" Doing so would merely subject me to **the why question** and the disclosure that, despite software's momentous societal impacts, I have regarded the word itself as, in successive stages, unserious, unenduring, effete, and -- bite your tongue, a *misnomer*. "Excuse me?"

Later I joined American Automation as VP of Marketing, where I labored for seven years in the rapidly advancing, software-rich world of "development systems" for embedded microprocessors. Throughout the 1980s, then, I was thoroughly habituated to neglecting the subject of this story, thus adding another decade's answer to **the why question**.

1990s

We come now to the last decade of the century -- the millennium, in fact, remember? Gradually there arose all those mystical decimalizings that infected people with collective expectations. Something more than the millennium was thought by some to be coming to an end on New Years Eve, 1999. In my case, however, there would be no apocolyptic nonsense looming, for I had begun a whole new career in 1991. In railroading.

Trains, people! I found a new technological religion and became baptized. For me it would mean nothing less than total immersion in a sacred river of safety critical engineering principles -- discovering, learning, and then influencing automatic train control systems.

The railroading industry was overdue for major innovations, and like every other realm of life in the 1990s, the trains of the world would experience radical changes and immeasurable benefits attributable to silicon and software. Thus, during what must be called my senior years, I have been deeply grateful for the opportunity to catch yet another technological wave. There would be, as the saying goes, new stories to tell my grandchildren.

The late Tom Sullivan ranked among the world renowned authorities in railroad signaling and control. I first met him at his office in New York City Transit in 1992. He became my mentor. Later he moved to California and became my best friend.

At an Equinox party in his hillside home, Tom Sullivan reminded me of something I had not thought about for a long time when he casually introduced me to his guests.

"Paul here is the guy who invented the word 'software'."

Excuse me. I think I have told all this before.

At that moment and for the first time, I became aware of a retrospective question...

Why would the person who coined the word 'software'
wait a half century before claiming it?

Sitting here fully alive in *The Software Age*, all my answers seem lame.

Nevertheless, be advised, I did *not* go to my grave without claiming the invention of the word 'software'. So there!

Part 8

Recommendations: Packing for Ego-Trips

Foremost, you must make a decision about your ego. Do you have one? Of course you do, but do you admit that you have an ego? While you're thinking about the answer to that question, permit me to remind you that without the sense of your own identity, either by self-perceptions or external representations, you might never enjoy the experience of what has been called an "ego-trip." That would be sad enough. Worse, without an ego, you might not ever do anything *worthwhile*. If you can bring yourself to agree with that, then you have already accepted my first recommendation, which is to go ahead and unabashedly acknowledge to yourself that you have an ego.

Whatever your age, whatever your station in life, you have already *done* plenty of worthwhile things. You have built or repaired something, you have said or taught something; likewise you have prevented or encouraged, written or drawn, you have probably invented -- *some*thing. You have been *compensated* for doing things, of course, but not always. More importantly, you have been *recognized* for doing things. But not always.

> "No matter," you shrug.
> "Matter!" I shout.

Oh sure, you can join the ranks of those who decline Oscars and Nobel Prizes, but look again. Are not such persons merely acting out an extreme manifestation of their own egos? Talk about self-reference: What an ironic ego-trip it must be to spurn worldwide acclaim!

Your own ego has taken a beating sometimes. Come on, admit it. Still, you realize that not *every*thing you have done deserves to be recognized. Over the years, you yourself may have *thought* that a particular thing you were doing deserved to be recognized -- at the time of the doing. Egos are like that. However, it has not always turned out that way, has it?

Just as often you have given little thought to something you once said, taught, prevented, encouraged, written, drawn, invented -- only to learn later that it did indeed turn out to be worthwhile. At that moment, [expletive deleted] you *know* you deserve to be recognized for it. I speak from experience here.

Ah, but maybe somebody else got recognized instead, even acclaimed. That must have been hard on your ego. Then too, sometimes *no*body gets recognized. Thus, nobody's ego gets the benefit -- at least from external sources. The question of who deserves to be recognized often comes down to a matter of firstness. Were you first? Go ahead and say so, but wait. How do you prove it?

> If you have invented something, then you know the problem well. You *thought* that your invention was worthwhile -- at the time you were doing the inventing. You didn't tell anybody, though. It was your secret. Time goes by

and -- poof! -- there it is on the market. "Hey," you exclaim to your friends and family. "That was my invention." Oh, right. If you say so.

> *Nota bene*: The best inventors must accept with equanimity that their best invention has already been invented by others -- or worse, that their best invention has been *obsoleted* by others.

Moreover, if you are a serious inventor, you must worry about the worst of all worlds: being forbidden to benefit commercially from your own invention. That's what other people's patents do. As a serious inventor, you know that a patent is a license to sue, and you sure as hell don't want to get sued for using your *own* invention. Thus, you will always apply for patents to secure for yourself the benefits of each invention as well as -- hoo-hah! -- to forbid others from doing so.

> Wrong! At least, not "always."

In the first place, not all applications result in patents. In the second place, not all patents are enforcable. The main reason is that during the years required for your patent application to be processed -- in secret -- other serious inventors were out there inventing away -- also in secret.

Those other inventors may or may not be applying for patents. Instead, they are routinely protecting themselves by doing what I most want to recommend here...

> Each is keeping a *contemporaneous holographic record.*

That means indelibly recording their work in a bound notebook on pre-numbered pages, dating each contiguous entry and embedding periodic signatures of comprehending witnesses. Such documents will forever establish the independence as well as the time-line of their own inventions.

When your patent finally issues, you might go ahead and sue those other inventors, though often as not, you will not prevail. Those others have routinely protected *them*selves by doing the same thing you are able to do to protect *your*self -- and without the trouble and expense of patent applications.

As in nature, where there is no predictive test of fitness, the worthwhileness of anything you do cannot be determined except retrospectively. If you want to claim credit for something worthwhile, you must plan ahead.

If you think that something you have just done will turn out to be worthwhile, then you can resort to an old-fashioned procedure -- you can write yourelf a registered letter, which you keep in a drawer unopened until some future historical moment when your effort has proven to be worthwhile and your ego will want you to stake your claim. Enough of those envelopes and you may lose track, so you will want to keep a separate index somewhere. But that's not solving the real problem, is it? The real problem is that you may *not* think that something you have done today will ever turn out to be worthwhile. You won't waste the postage, and that will be the end of it.

> Excuse the personal references here, but since my children were old enough to write, I have repeatedly urged each one to keep a diary. That they did not accept my advice is doubtless attributable in part to my not doing so myself.

Well known to my children, however, are the stacks of engineering notebooks now buried in the archives of all my former employers, each intended to protect the respective enterprises against "interference claims" by outside patent holders. While not exactly the same as a diary, the notebooks followed a protocol that mandates recording all events of the day (phone calls, meetings, conversations -- even dental appointments and grocery lists) as well as technical content (diagrams, ideas, analysis, observations, data, calculations). Like all my patents, however, those notebooks are the property of my former employers -- emphasis on "former" -- and thus not conveniently accessible to me.

Augmenting those records during the years of the "paper deluge" were my chronological "day-files" -- copies of all outgoing correspondence, including personal letters. These files over time became immense and, being in loose-leaf form, piles of pages got selectively purged on residential moving days.

> One letter, I recall, went to a friend in early 1971. It described the coining of a new word. I had just been given overview responsibility for XEG (Xerox Education Group, which included R. R. Bowker, publisher of *Books in Print*, and American Education Press, publisher of *My Weekly Reader*). That year, XEG funded an underwriting grant to PBS for *Sesame Street*. I watched an episode and got to pondering the implications of what I saw.
>
> The word 'edutainment' detonated inside my cranium. Later I went around using that neologism in talks at universities and management groups extolling the power of media to teach. Now, after more than thirty years, the term 'edutainment' has achieved currency but does not yet appear in the *OED*. When it does, I would be pleased to have my name next to it; however, I cannot provide the requisite documentation, since that contemporaneous letter has long ago been discarded from my day-file.

Each day, write something in your diary. That is my recommendation to you. Over the years, that is what I have told anybody who will listen and others, for example students, who had no choice. "Write each day -- not because you have something to say," I like to say, "but because it is *that* day." If you cannot think if any significant events, then write that down. Enough of those days, though, and your diary is telling you to get a life.

> Internet technology has come along to offer a convenient alternative for doing your diarizing. You can send an e-mail message each day to some number of correspondents -- a circle of archiving partners, so to say. Subsequent retrieval of identical content from multiple computers, with consistent date-stamping, will surely authenticate your primacy against even the most suspicious challenger.

To be sure, you won't know *a priori* the eventual value of your daily entries, but you have laid the foundation -- you have put in place the time-line -- for authenticating your claims to anything you will ever do that subsequently reveals itself to the world as being worthwhile -- including the coining of an important word. The payoff for that particular entry then can be immense, and your ego will be the beneficiary. Not doing so -- not planning ahead -- can result in painful lamentations as this memoir demonstrates.

1953 "Software" was merely a prank.

Part 9

Softword:Provenance for the Word 'Software'

Appendices

ISBN 1-58922-233-4

Your comments are invited. Please write to paul@niquette.com

<< Previous	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9	Next Part >>
Home Page	Softword: Table of Contents	Top of Article

Appendix A -- Dictionary Entries

Oxford English Dictionary

Oxford University Press, 1999

software

[f. soft*a.* + ware*n.*^{3}, after hardware 1c.]

1.*Computers*.**a.** The programs and procedures required to enable a computer to perform a specific task, as opposed to the physical components of the system (see also quot. 1961).**b.***esp.* The body of system programs, including compilers and library routines, required for the operation of a particular computer and often provided by the manufacturer, as opposed to program material provided by a user for a specific task.

In early use, the word was interpreted widely to include program material written by a user, as well as systems programs, and also occas. the cards and tapes by means of which programs and data are read into the system. Popular usage, as represented by sense 2, is freq. wider in meaning than the current more restrictive technical usage (sense b).

"**1960***Communications Assoc. Computing Machinery* June 381 Nearly every manufacturer is claiming compatibility with all other equipment via such software as Cobol.""""**1961***Computer Bull.* June 42 The programming expertise, or 'software', that is at the disposal of the computer user comprises expert advice on all matters of machine code programming, comprehensive libraries of subroutines for all purposes, and the pegasus/sirius scientific autocode."**1962**D". S. Halacy*Computers* iii. 54 Punched cards, which fall into the category called computer 'software' are cheap, flexible, and compatible with many types of equipment."**1964***Observer* 13 Dec. 1/1 The toughest problem was the 'software'—particularly the 'supervisory programme', the complex instructions which enable the machine to handle many tasks simultaneously.""**1965**H"ollingdale & Tootill*Electronic Computers* 192 The cost of developing and making the computer itself (the *hardware*) is matched by the cost of making programming schemes for it (often, regrettably, termed *software*)."**1966***New Scientist* 25 Aug. 433/3 The cost of providing 'software'—the programmes for operating the computer on a wide range of problems—is enormous."+ The user needs to find the bureau which has the appropriate software for his problems."**1967**C"ox & Grose*Organization & Handling Bibl. Rec. by Computer* 1 About three years ago, it became clear+that the computer software which was provided and maintained by the manufacturers was not suited to some of the problems of handling and processing large files of data."**1969**P". Dickinson*Pride of Heroes* 187 A rather wet young man who sells software for computers."**1971**B". de Ferranti*Living with Computer* 89 *Software*, all computer programs, or that part of a computer system that is not hardware."**1972***Computer Bull.* XVI. 85/1 In those days [*sc*. 1966] the term 'software' was still thought rather disreputable, and the concept was probably thought rather vague."+ More recently, 'software' has become more particularised and often seems to refer to what we might call 'system software', that is, excluding any programs written for specific applications.+ Thus we have 'software packages' and 'application packages', and people who write software consider themselves superior to mere programmers."**1977**K". Heggstad in P. G. J. van Sterkenburg et al. *Lexicologie* 163 The unit price of hardware is going down.+ On the other hand software costs are rising equally dramatically."**1978**J". McNeil*Consultant* i. 30 Hardware is what you can touch—the actual computer, all its peripheral devices.+ Without software all that is quite useless.+ Software, computer programs—they're the same thing.+ My software staff are very strictly monitored.

2.*transf.* and *fig.*

“**1963***Flight International* LXXXIII. 186/1 To get at the total commitment one has to consider the ‘software’ aspect very closely: for every controller at the scope there may need to be five in the background.”“**1966***National Observer* (U.S.) 21 Feb. 8/3 This deal”+is the latest+in a series of corporate marriages combining+‘the software and the hardware’ of education.“**1967***Punch* 24 May 770/3 This documentary was a refreshing change from most space-age reportage, dealing sympathetically with the families of the astronauts living outside the perimeter fence of the Manned Spacecraft Centre in Texas: the software rather than the hardware.”“**1969***Guardian* 29 Mar. 4/8 The ‘Talking Page’ ”+is+being launched with a mass of matching software—a maths course, a reading course, an English course for immigrants.“**1978***Gramophone* June 136/3 They [*sc*. players for digitally recorded discs] will be usable with normal stereo amplifiers and speakers but, of course, they will be incompatible with existing software (records and cassettes).”“**1979***Observer* 11 Nov. 33/2 It was phrased in terms of Israel giving the United States ‘software’—a more flexible attitude on the Middle East—in return for ‘hardware’—arms and military equipment.”

3. Special Combs.: **software engineering**, the professional development, production, and management of system software; so **software engineer**; **software house**, a company that specializes in producing and testing software; also *fig.*

“**1969**Naur & Randell*Software Engin.* (NATO) 81 Is it possible to have software engineers in the numbers in which we need them, without formal software engineering education?”“**1979**J”ensen & Tonies*Software Engin.* 14 The software engineer is not a theoretician as is the computer scientist.

“**1969** (*title*) Software engineering; report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October, 1968.”“**1973**K”. W. Morton in F. L. Bauer *Adv. Course Software Engin.* i. A. 4 When we sit down at a console to write an Algol program, it is software engineering which determines how easy it is to achieve this end.“**1982**I”. Sommerville*Software Engin.* i. 3 Software engineering is now maturing into a fully fledged discipline.

“**1969***New Scientist* 6 Nov. 285/1 Today there are just over 2000 software houses throughout the world, mostly in America.”“**1982***Listener* 23–30 Dec. 31/1 If the world's wealth is maximised by specialisation, Britain should become its ‘software house’.”

Definitions from Selected Dictionaries

It is often forgotten that dictionaries are artificial repositories, put together well after the languages they define. The roots of language are irrational and of a magical nature.
-- Jorge Luis Borges, *El Otro, el Mismo*

software n (1960): something used or associated with and usually contrasted with hardware: as the entire set of programs, procedures, and related documentation associated with a system and especially a computer system; specif: computer programs.

-- *Merriam-Webster's Collegiate*

software n. Written or printed data, such as programs, routines, and symbolic languages, essential to the operation of computers.

-- *The American Heritage Dictionary of the English Language*
Houghton Mifflin Company 1981

soft•ware n.. 1. Computers.the programs used to direct the operation of a computer 2. anything that is not hardware but is used with hardware... Computer programs; also called "applications."

-- *Information Please*

software n. the programs that are used in a computer system (eg operating systems, and applications programs such as word-processing or database programs).

-- allwords.com

software n. 1966 Computer Science: The programs, routines, and symbolic languages that control the functioning of the hardware and direct its operation.

-- *The American Heritage Dictionary of the English Language*
Third Edition. 1996

software n. the instructions which control what a computer does; computer programs

-- Cambridge University Press 2000

soft·ware n. programs and applications for computer: computer programs and applications, such as word processing or database packages, that can be run on a particular computer system. [Mid-19th century. Originally, in plural, "soft goods." The modern sense dates from the mid-20th century.]

-- Encarta® *World English Dictionar*
North American Edition © 1999-2000 Microsoft Corporation.

software n. the collection of programs loaded externally which cause a computer to perform a desired operation or series of operations (opposed to hardware).

-- *The Macquarie Dictionary*

software n. any of the languages or programs, or instructions for using these, that are written for and used with a computer.

-- *Wordsmyth Educational Dictionary*

software n. Computers The programs used to direct the operation of a computer, as well as documentation giving instruction on how to use them. 2. Anything that is not hardware but is used with hardware...(1955-1960)

-- *Random House Dictionary of the English Language*
Second Edition, Unabridged 1987

soft wares = Dry goods. Dry goods Commercial -- Chiefly U.S. Textile fabrics, cottons, woolens, linens, silks, laces, etc. -- in distinction from hardware, jewelry, groceries, etc.

-- *Websters New International Dictionary*
Second Edition, Unabridged 1954
{Return to citation in Part 0}

<< **Previous**	**0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**	**Next Part** >>
Home Page	**Softword: Table of Contents**	**Top of Article**

Appendix B -- Challenger References

Update in April 2013:
Discovery of a Paper Published in 1956 Containing the Word *Software*

Discovery of a Paper Published in 1956 Containing the Word *Software*

Here is an excerpt from the Wikipedia page for John Tukey (with emphasis **added**)...

> The term "software", which Paul Niquette claims he coined in 1953,[6] was first used in print by Tukey in a 1958 article in *American Mathematical Monthly*, and thus some attribute the term to him.[7] That 1958 **first use and attribution is questionable**, given that the *Proceedings of the Second National Symposium on Quality Control and Reliability in Electronics: Washington, D.C., January 9-10, 1956*, as accessible on Hathi Trust, contains an occurrence of the term *software*.[8]

…which was brought to my attention by Dick Weaver in 2013, in a message that...

- took note of my efforts to document pre-1958 uses of the word *software* as described in *Softword* Part 6;
- inquired about whether, by any chance, the author of the 1956 reference, Richard R. Carhart, might have been one of my students prior to 1956; and...
- provided a link to the *Proceedings of the Second National Symposium on Quality Control and Reliability in Electronics: Washington, D.C., January 9-10, 1956*.

My records do not include a student named Richard R. Carhart, which puts in doubt that he first heard or read the word *software* directly from me before 1956. The name Richard R. Carhart does appear in several web-pages, including this entry in Microsoft Academic Search, but so far my efforts to contact him have been unavailing.

Obituary for John Wilder Tukey

July 28, 2000

John Tukey, 85, Statistician Who Coined 2
Crucial Words

By David Leonhardt

John Wilder Tukey, one of the most influential statisticians of the last 50 years and a wide-ranging thinker credited with inventing the word "software," died on Wednesday in New Brunswick, N.J. He was 85.

The cause was a heart attack after a short illness, said Phyllis Anscombe, his sister-in-law.

Mr. Tukey developed important theories about how to analyze data and compute series of numbers quickly. He spent decades as both a professor at Princeton University and a researcher at AT&T's Bell Laboratories, and his ideas continue to be a part of both doctoral statistics courses and high school math classes. In 1973, President Richard M. Nixon awarded him the National Medal of Science.

But Mr. Tukey frequently ventured outside of the academy as well, working as a consultant to the government and corporations and taking part in social debates.

In the 1950's, he criticized Alfred C. Kinsey's research on sexual behavior. In the 1970's, he was chairman of a research committee that warned that aerosol spray cans damaged the ozone layer. More recently, he recommended that the 1990 Census be adjusted by using statistical formulas in order to count poor urban residents whom he believed it had missed.

"The best thing about being a statistician," Mr. Tukey once told a colleague, "is that you get to play in everyone's backyard."

An intense man who liked to argue and was fond of helping other researchers, Mr. Tukey was also an amateur linguist who made significant contributions to the language of modern times. In a 1958 article in American Mathematical Monthly, he became the first person to define the programs on which electronic calculators ran, said Fred R. Shapiro, a librarian at Yale Law School who is editing a book on the origin of terms. Three decades before the founding of Microsoft, Mr. Tukey saw that "software," as he called it, was gaining prominence.
"Today," he wrote at the time, it is "at least as important" as the "'hardware' of tubes, transistors, wires, tapes and the like."

Twelve years earlier, while working at Bell Laboratories, he had coined the term "bit," an abbreviation of "binary digit" that described the 1's and 0's that are the basis of computer programs.

Both words caught on, to the chagrin of some computer scientists who saw Mr. Tukey as an outsider. "Not everyone was happy that he was naming things in their field," said Steven M. Schultz, a spokesman for Princeton.

Mr. Tukey had no immediate survivors. His wife of 48 years, Elizabeth Rapp Tukey, an antiques appraiser and preservation activist, died in 1998.

Mr. Tukey was born in 1915 in New Bedford, a fishing town on the southern coast of Massachusetts, and was the only child of Ralph H. Tukey and Adah Tasker Tukey. His mother was the valedictorian of the class of 1898 at Bates College in Lewiston, Me., and her closest competition was her eventual husband, who became the salutatorian. Classmates referred to them as the couple most likely to give birth to a genius, said Marc G. Glass, a Bates spokesman.

The elder Mr. Tukey became a Latin teacher at New Bedford's high school, but, because of a rule barring spouses from teaching at the school, Mrs. Tukey was a private tutor, Mrs. Anscombe said. Mrs. Tukey's main pupil became her son, who attended regular classes only for special subjects like French. "They were afraid that if he went to school, he'd get lazy," said Howard Wainer, a friend and former student of John Tukey's.

In 1936, Mr. Tukey graduated from nearby Brown University with a bachelor's degree in chemistry, and in the next three years earned three graduate degrees, one in chemistry at Brown and two in mathematics at Princeton, where he would spend the rest of his career. At the age of 35, he became a full professor, and in 1965 he became the founding chairman of Princeton's statistics department.

Mr. Tukey worked for the United States government during World War II. Friends said he did not discuss the details of his projects, but Mrs. Anscombe said he helped design the U-2 spy plane.

In later years, much of his important work came in a field that statisticians call robust analysis, which allows researchers to devise credible conclusions even when the data with which they are working are flawed. In 1970, Mr. Tukey published "Exploratory Data Analysis," which gave mathematicians new ways to analyze and present data clearly.

One of those tools, the stem-and-leaf display, continues to be part of many high school curriculums. Using it, students arrange a series of data points in a series of simple rows and columns and can then make judgments about what techniques, like calculating the average or median, would allow them to analyze the information intelligently.

That display was typical of Mr. Tukey's belief that mathematicians, professional or amateur, should often start with their data and then look for a theorem, rather than vice versa, said Mr. Wainer, who is now the principal research scientist at the Educational Testing Service.

"He legitimized that, because he wasn't doing it because he wasn't good at math," Mr. Wainer said. "He was doing it because it was the right thing to do."

Along with another scientist, James Cooley, Mr. Tukey also developed the Fast Fourier Transform, an algorithm with wide application to the physical sciences. It helps astronomers, for example, determine the spectrum of light coming from a star more quickly than previously possible.

As his career progressed, he also became a hub for other scientists. He was part of a group of Princeton professors that gathered regularly and included Lyman Spitzer Jr., who inspired the Hubble Space Telescope. Mr. Tukey also persuaded a group of the nation's top statisticians to spend a year at Princeton in the early 1970's working together on robust analysis problems, said David C. Hoaglin, a former student of Mr. Tukey.

Mr. Tukey was a consultant to the Educational Testing Service, the Xerox Corporation and Merck & Company. From 1960 to 1980, he helped design the polls that the NBC television network used to predict and analyze elections.

His first brush with publicity came in 1950, when the National Research Council appointed him to a committee to evaluate the Kinsey Report, which shocked many Americans by describing the country's sexual habits as far more diverse than had been thought. From their first meeting, when Mr. Kinsey told Mr. Tukey to stop singing a Gilbert and Sullivan tune aloud while working, the two men clashed, according to "Alfred C. Kinsey," a biography by James H. Jones.

In a series of meetings over two years, Mr. Kinsey vigorously defended his work, which Mr. Tukey believed was seriously flawed, relying on a sample of people who knew each other. Mr. Tukey said a random selection of three people would have been better than a group of 300 chosen by Mr. Kinsey.

{Return to citation in Part 6}

OED entry for "bit."

1. A unit of information derived from a choice between two equally probable alternatives or ‘events’; such a unit stored electronically in a computer.

1948 C. E. Shannon in *Bell Syst. Techn. Jrnl.* July 380 The choice of a logarithmic base corresponds to the choice of a unit for measuring information. If the base 2 is used the resulting units may be called binary digits, or more briefly *bits*, a word suggested by J. W. Tukey. **1952***Sci. Amer.* Sept. 135/1 It is almost certain that ‘bit’ will become common parlance in the field of information, as ‘horsepower’ is in the motor field. **1957***New Scientist* 9 May 14/1 One ‘bit’ is the smallest amount of data which can exist, and corresponds to the answer to a yes-or-no question. On this basis, a decimal numeral can be described with four bits and an alphabetic letter with five.+ Existing electronic computers can store, in their normal memories, up to about one million bits.

OED entry for "byte."

A group of eight consecutive bits operated on as a unit in a computer.

1964 Blaauw & Brooks in *IBM Systems Jrnl.* III. 122 An 8-bit unit of information is fundamental to most of the formats [of the System/360]. A consecutive group of *n* such units constitutes a field of length *n*. Fixed-length fields of length one, two, four, and eight are termed bytes, halfwords, words, and double words respectively. **1964** *IBM Jrnl. Res. & Developm.* VIII. 97/1 When a byte of data appears from an I/O device, the CPU is seized, dumped, used and restored. **1967** P. A. Stark*Digital Computer Programming* xix. 351 The normal operations in fixed point are done on four bytes at a time. **1968** *Dataweek* 24 Jan. 1/1 Tape reading and writing is at from 34,160 to 192,000 bytes per second.

<< Previous	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9	Next Part >>
Home Page	Softword: Table of Contents	Top of Article

Appendix C -- Selected Correspondence

The Verb "Sofware"

Excerpts from e-mail messages in 2000 from Emil Borgers, an eminent figure in programming from the fifties onward...

Borgers: Programmers in 1957 were never called 'software engineers' and I corrected anyone who tried. I remember being insulted by its use --- an annoyance really.

The common phrase 'to software around it' was how the term was used the most, much to my dismay -- basically, by engineers who had no more room for components (or money in the design budget) and wanted to slip the problem into someone else's domain (mine). Today, with processing speed as it is, 'software-arounding' things is normal but in those days it was not often a wise choice.

The "Ralliac"

In 2006, another colleague from the fifties responded to my request for recollections about adventures with early computers. Bill Paine was the programmer who wrote the bit-by-bit simulator for the RW-300 on the Univac 1103A and then became a regional sales manager for TRWP.

Paine: In the sifting through the shards of early 'computerdom' a modern archeologist finds...a lot of dust. There must be a better use of time in one's final remaining years! Churchill wrote his *History of the English Speaking People* in his later years.

> **PN:** How much do you recall about 1957 and the first international sports car rally in the U.S. -- and the "Ralliac"?

Paine: The idea for entering the rally was mine, and I invited you to be my partner. The Ralliac was conceived by you when I explained the need for mastering on-board real-time calculations required in a sports car rally. The main goal was to finish an initially unknown course, hitting unknown check points at the exact times specified by the rally designers. Therefore it was critically important that the rally team (driver and navigator) have accurate real-time information on time and distance traveled.

The Ralliac was designed to produce the critical mileage information using pulses generated by a magnet on a wheel passing the 'read head' mounted closed by. Precise timing would be supplied by clock circuits within the Ralliac.

Sports car rallies were international events in those days. They often lasted 24 hours and represented a total distance to be traveled of many hundreds of miles. There would be dozens of contestants. Counting the planning and checkpoint staffing for the rally, there must have been many other people spending a great deal of time in addition to the rally contestants.

Our planning for the first application of the Ralliac included its installation in a new car donated by a local dealer...

> **PN:** As the "hardware" guy, my assignment called for me to provide a demonstration at a Beverly Hills luncheon sponsored by a certain Renault dealer. All I had at that time was one printed circuit board. It was populated with advanced electronic components like diodes and transistors and was just small enough to fit in your jacket pocket.
>
> In his introduction to your presentation, the owner of the dealership exulted about cars ("the era of the small car is upon us") and about new electronic technologies (using the not uncommon mispronunciation "digitile computors"). At the appropriate moment in your presentation, you passed the board around the table. That was all we needed. The subject of your "software" never got mentioned. The same afternoon, we were handed the keys to a brand new sedan.

Paine: We never entered the rally due to circumstances beyond our control, as I recall.

> **PN:** Right you are, Bill. The rally was suddenly cancelled by the sponsor. European contestants were caught in mid-ocean with their sports cars covered with cosmoline.
>
> After several week-ends of frantic soldering and coding, you and I suffered the instant obsolescence of the Ralliac. And I shall never forget the day we had to return the no longer new Renault to the dealer.

The Confusing Term: "Programming"

Exerpt from e-mail messages in 2006 from Stu Schy, colleague at TRW Computers, commenting on passage in Part 2: "The word 'programming' was serviceable enough, of course. Renaming it 'software' did more to assure my reputation as an eccentric than to illuminate computer technology."

Schy: In the early 1960s, a group of us from TRW Computers attended a meeting at CBS headquarters in New York prior to the contract signing for an automatic switching system for the CBS network using the RW-300. There were a number of CBS VPs there. One of the CBS people asked "What do we do if we change from a CBS type fade to an NBC type fade?" This referred to how running overtime was handled. I answered, "We can handle that with a small change in the program (meaning the timing subroutine.) At which, one of the men at one end of the table took the cigar out of his mouth, stood up, pounded his fist on the table and said, "No one is messing around with my programming!"

He was the CBS VP for Programming. This was one of a number of incidents that taught us the need to hire people from the industries we hoped to serve if we wanted to speak their language. We did get the contract. Some time later, a CBS technician said to me "Hey, you bastard, you cost me my job! Now, how about giving me one?" I said, "That's fair." and we hired him. After a few months, he used the expertise he learned at TRWC to get a much better job at CBS as Manager of Computer Operations.

My Mentor: Dan Gerlough

While conducting my researches for this memoir, I received an e-mail message on March 24, 2001 from Walter F. Bauer, a renown figure in the early history of computers. His recollections about Dan Gerlough stirred up my own memories. I replied the same day. Let this exchange serve as a tribute to Dan and to his enduring contributions to traffic engineering.

Bauer: In reading [the draft], I couldn't help recalling my contacts with Dan Gerlough. In 1960 Dean Wooldridge [the W in TRW] got the idea to start a traffic control business (project). I was chosen to start it.

> **PN:** Coincidence Alert: During that time period, Walt, I was also with TRW, on a consulting assignment in air traffic control at the fledgling FAA at the National Aviation Facility Experimental Center (NAFEC) in Atlantic City, so I did not know about the [vehicular traffic control] initiative.

Bauer: The idea was to apply electronics and modern analysis to the emerging traffic problem, especially the freeways. I got a basic understanding of what was going on in traffic control technology, travelling around the country and talking to the cognoscenti.

> **PN:** Ironically, just last month [April, 2001], I attended a "brown-bag" luncheon in San Francicso put on by one of my colleagues at Booz Allen Hamilton. Our firm has a consulting contract with Caltrans and other highway jurisdictions for -- well, suffice it to say, I have no doubt that you would find the technical means for characterizing traffic congestion, and the remedies are all familiar enough. The Law of Unintended Consequences prevails universally. There is a tangle of conflicting interests that impede solutions, if indeed solutions there be (apart from bicycles, of course).

Bauer: Not long after starting, I heard about Dan and his project at UCLA. I convinced him to head up the traffic control group in my department at TRW. We got some prestigious traffic control contracts, and developed a road sensor.

> **PN:** At UCLA's ITTE in 1953, a couple of us, with Dan Gerlough's approval and mentoring, invented an electric wheel sensor, using a rubberized tape with sinuous metal conductors inside, to replace the pneumatic tube contraptions of that time. We placed them in pairs across traffic lanes, separated by a yard, and, with an

electromechanical timing device, we were able to classify speeds, thus solving what I called the "density-lock deception" [flow-rates measured with single-point detectors will *decrease* with congestion tricking the analysis into thinking congestion is clearing out]. My guess is that your sensor was the "embedded loop"; if so, it is still in use today -- also deployed in pairs, but, alas, often not connected to any recording devices, due to budget constraints.

Bauer: But our best claim to fame was applying the RW-300 for intersection traffic control. The computer handled some 50 intersections around Dodger Stadium. It was the very first time a computer had ever been employed for automatic traffic control! I wonder if any historian would be interested in that.

PN: One quasi-historian is interested; that's for sure. More coincidence, Walt. I was involved about a year later with an awkward, non-computerized system at Scantlin for controlling clusters of intersections around the Los Angeles Coliseum.

Bauer: Post Script: I remember many discussions in 1961 with traffic engineers on the subject of on-ramp metering. We were ready to install, but they rejected it out of hand. It wasn't until about 10 years later that on-ramp metering became a reality. And now it's commonplace. The idea, as you well know, is that when density passes a critical stage, volume drops off precipitously.

PN: Yes, I do know about this idea, which has its roots as follows:

- In mid-1953 some of us at ITTE, with Dan Gerlough's approval and mentoring, did a study of afternoon congestion on the UCLA campus -- a time when departing vehicles should have been diverging and therefore more or less free-flowing. Our hypothesis was that a goodly (badly?) number of cars parked in lots at the South end travelled through the campus to exit North on Sunset Blvd., and vice versa, North parkers exiting southbound onto Westwood Blvd. We hired a bunch of "student workers" at $1.25 per hour to dab poster paint onto bumpers of parked cars, color coded by lot. In the late afternoon, we stood around the campus, clicking our Veeder-Root counters and -- well, the rest is history. The remedy was -- and is -- the one-way street system on the UCLA campus.

- In late-1953, some of us at ITTE, with Dan Gerlough's approval and mentoring, did a study of the Hollywood Freeway, which suffered anomalous congestion in the morning. You will recall that not all freeway entrances were on the right side in those days. Our hypothesis was that a badly number of commuters came onto the freeway -- on the left side -- at Barham Blvd and upon seeing congestion ahead, proceeded to change lanes to the right in order to get off at the next exit ramp (Silverlake), inflicting thereby an increase in travel time upon every vehicle behind them -- about 1.75 seconds each, as I recall. Poster paint was out of the question. Instead, we used Tel-Audograph recorders (magnetic tape? -- what's that?) to record license numbers, along with time codes at Barham and at Silverlake. Then we punched up Hollerith cards for the SWAC and -- well, the rest is history. The remedy for that time, by the way, was the extreme in vehicle metering: saw-horses across the on-ramp at Barham every morning.

- In early-1954, I began my work on density lock with Dan Gerlough's approval and mentoring, which culminated in my undergraduate thesis paper in 1955, which was based on controlled experiments on Hildegard Blvd. and won the Deans Award (ahem), entitled "Experiments in Spacing of Stopped Vehicles for Improving Linear Flow." Hey, I even got to do some crude modeling on the SWAC, using the line printer for a primitive "graphical display" showing platoons of vehicles, each depicted with a two-digit code representing their respective speeds. Hot stuff, that.
- In late 1954, I began promoting the idea of "vehicle metering" in speeches and campus forums, repeatedly refuting doubters ("It makes no sense to stop cars getting onto the freeway!") using a slogan, "If you don't stop them before they get on, density lock will make the freeway into a parking lot." Dan Gerlough bought into my argument, but, like so many other innovations for which we were advocating (seatbelt interlock, radar speed meters, high intensity runway lights), metering signals would take years -- and your imprimatur along with Dan's, apparently -- before acceptance.

Bauer: Dan Gerlough was one of my favorite people. I remember well his easy, avuncular manner.

> **PN:** So do I. At that "brown bag" on traffic congestion last month, Walt, I saw a slide depicting a venerable scattergram -- that plotting of double-valued flow-rate versus speed (a horizontal "V" -- remember?) and got tears in my eyes.

Exchange with Ray Stanish

One student at Los Angeles Tech and later a close friend, Rig Currie, did act on my behalf by writing an e-mail message to another student, Ray Stanish. Nice try, Rig.

October 27, 2000

Dear Paul;

All the time I worked at TRW, I don't recall hearing the term "software."

I left TRW to become a Professional Public Speaker. I started out with the talk "Atomic Energy - Peasant Style" but soon thereafter developed one I called "Giant Nincompoops" -- about computers obviously. I presented that talk off and on for 10 or 15 years, and I don't recall during all that time ever using the term
"software" in my talks, so I couldn't have been aware of it.

So, I have no idea when I became aware of it or the source.

Sorry, I can't help you, Paul. I would've been glad to if I could. I always had high regard for your brains and your 'way-out-in-front knowledge of computers.

It's been a pleasure to have known you and to have learned from you.

Your friend,
Ray

October 29, 2000

Thanks for your message, Ray.

Perhaps you do remember that in 1957, TRWP (pronounced "twerp") was solemnly engaged in launching a new business development that called for marketing unfamiliar technology in a risk-averse business environment -- the earliest applications of small, primitive digital computers ("digitile computors") that would be put into service exerting absolute real-time control over safety-critical systems, including chemical factories, oil refineries, and nuclear power plants. "Closing the Loop," read the headlines.

Nothing to kid around about. Soft things need not apply.

Ten years later, Ray, I had the privilege of attending a performance of "Giant Nincompoops" at a management club banquet at Scientific Data Systems. I sat crimson-faced in the audience as the speaker singled me out for recognition as the pedagogical source (LA Tech, remember?) for the material being lampooned -- but not the comedic inspiration for the lampooning.
And no, I don't think the talk included so much as one utterance of the word 'software,' which for general audiences -- even in 1967 -- would have necessitated a pause for etymology (see "Cultured Laughter").

My own laugh-line in those days was not suitable for general audiences.

> Imagine your wife complaining to her mother, "All he ever talks about is software."

My memories are vivid for that time, Ray. Do you remember the "Million Dollar Order Party"? And the song you wrote based on IBM's company anthem?

> We are loyal and true, Ramo-Woo
> For together we stand
> To fight Remington Rand,
> IBM and Bendix, too.

Other than in my class, there were really not many opportunities for you to have heard me say the word 'software'. In the middle of my tenure, just before TRWP moved to Beverly Hills, I broke daily contact with you and others and went off to consult for the fledgling FAA in Atlantic City, NJ, applying the RW-300 in -- gasp! -- air traffic control.

Well, while consulting for the FAA, I also wrote simulation software for the IBM 709 and commented with a straight face...

> "After last night, I don't seem to have any more problems with my software."

Being smirked at by IBM programmers in their Florsheims and striped ties kept me from smirking.

Best regards,
Paul

Computer Languages (Plural).

Commenting on a draft of Softword, a historian of computer languages at Murdoch University, Diarmuid Pigott wrote:

Pigott: Just read your book on the word "software," and was very excited by it: I shall make sure that everyone learns of your coinage, and correct those who remain in ignorance.

When ideas are named, as you no doubt realize, they act as seed crystals for inspiration, and only when something has a name can it become the subject of study, thought, and discipline. So if you hadn't come up with the name, all would have been different. I also think that it was important that it be a neologism because some of the stupid debates about the meaning of "knowledge" and "information" would not have gotten off the ground if the term had not been clearly different (as in "real-information" vs. "computing information").

I have spent some considerable time over the last few years trying to work out the origin of the word "language" in the sense of instructions for computers. The word programming is older, obviously coming from optimization and linear programming, but the curious sense of "language" used -- which has gotten all sorts of people into all sorts of knots by making them think of human languages rather than thinking of machine codes and symbologies -- derives from a minor, restricted use that was formerly considered arcane.

What we know of it is:

- Not used by Aitken, Booth, Church, Curry, Goedel, Laning, Lovelace, Post, Turing, Von Neumann, Wilkes. They use "system," "code," "instructions";
- Prior use of the term to describe closed system of word – mathematical language, engineering language;
- Use in this sense by Backus 1954 "Can a machine translate a sufficiently rich mathematical language into a sufficiently economical program?";
- Prior use as system of signs (semiotic use);
- Used in modern sense by 1958 (AlGOL, IAL, JOVIAL, CODASYL etc.);
- Standard term for programming system by 1962.

The earliest usage I can find is Edmund Berkeley 1949 "Since it is not always easy to think, men have given much attention to devices for making thinking easier. They have worked out many systems for handling information, which we often call languages. Some languages are very complete and versatile and of great importance. Others cover only a narrow field—such as numbers alone—but in this field they may be remarkably efficient. Just what is a language? Every language is both a scheme for expressing meanings and physical equipment that can be handled."

I would love for you to draw on your rich storehouse of memories and help me unravel this problem! Thanks anyway for your most illuminating work.

> The mental exercise to which I was invited by this message proved unavailing, except that I am now all the more sympathetic to the difficulty faced by my friends and family in trying to respond to my queries about the word 'software' (when did you first hear it? learn its meaning? from whom?)

Thus, a lifetime being subjected to the use of the word 'language' in diverse contexts (the language of aviation, culinary language, the language of birds, the language of logic, of love, of storms, medical language, cowboy lingo, diplomatic language) has obliterated my recollections of its earliest application in the realm of computers.

The word 'lan·guage' has made a long but, I suspect, rapid metaphorical journey from its literal origins...

> Etymology: Middle English, from Old French *langage*, from *langue*, tongue, from Latin *lingua*.

...to its appropriation for entities-without-tongues.

> 1a. Communication of thoughts and feelings through a system of arbitrary signals, such as voice sounds, gestures, or written symbols. b. Such a system including its rules for combining its components, such as words. c. Such a system as used by a nation, people, or other distinct community; often contrasted with dialect. 2a. A system of signs, symbols, gestures, or rules used in communicating: the language of algebra. b. **Computer Science A system of symbols and rules used for communication with or between computers** [emphasis added]. 3. Body language; kinesics. 4. The special vocabulary and usages of a scientific, professional, or other group: “his total mastery of screen language—camera placement, editing—and his handling of actors” (Jack Kroll). 5. A characteristic style of speech or writing: Shakespearean language. 6. A particular manner of expression: profane language; persuasive language. 7. The manner or means of communication between living creatures other than humans: the language of dolphins. 8. Verbal communication as a subject of study. 9. The wording of a legal document or statute as distinct from the spirit.

-- *American Heritage Dictionary*

Apparently just about any form of information exchange, with or without the use of a literal *tongue*, can be a language so long as sender and receiver agree on a protocol. Now, according to my way of thinking, that incudes every form of expression except modern art. But then, at a typical exhibition, I may be merely a defective receiver surrounded by exceptionally gifted senders using languages that I am unable to fathom.

Appendix D -- End Notes

Number of Words in the English Language

Each English word encodes its concept -- concepts, plural -- using 26 discrete symbols, much like a decimal number encodes a quantity using ten discrete symbols. Indeed, it is possible to represent the number of subatomic particles in the Milky Way Galaxy with a string of fewer than 38 cyphers, each confined to only ten different shapes. Likewise, with variable-length strings of letters, each confined to only 26 different shapes, English can represent all 38,000 words, which is quite a remarkable feat, considering that other exceptionally popular languages (need I say Chinese?) require the use of almost that many written -- drawn -- symbols.

In early 2001, Fred Shapiro, Associate Librarian for Public Services and Lecturer in Legal Research at Yale Law School expressed concern that my estimate of 38,000 is too low, suggesting that the *Oxford English Dictionary* lists many times that amount. I shall take his word for that.

The estimate of 38,000 English words stuck in my memory from some long time ago (38,006, perhaps, since I have added a half-dozen myself). Upon receiving Shapiro's query, though, I took down from my shelf *The American Heritage Dictionary of the English Language*, which is not the OED but one I happen to like a whole lot. Now, there are 1,491 pages with definitions on them. I did a quick count of the words defined on a couple of randomly selected pages and got an estimate of 25.5 words. You can do the indicated arithmetic, of course, but I'll save you the trouble: 1,491 times 25.5 equals 38,020 words. In the present context, I consider that number to be quite generous, since it includes entries like Aalborg and Zola, which ought to be excluded since they do not qualify as invented words capable of offsetting the paucity being lamented here.

Mr. Shapiro expressed the belief to me that William Shakespeare used 33,000 separately definable words. I shall take his word for that, too. Assuming the Bard used all the English words available in his time, then people have succeeded in adding only about one word-per-month to the language throughout the past four centuries. One of them, as I have noted elsewhere, is "byte."

Another, of course, is 'software.'

{Return to citation in Part 0}

Run vs Set

Once, I needed to select a clickable word to appear on a computer screen, and "run" seemed to fit the requirement. I looked up "run" in my favorite dictionary, *The American Heritage Dictionary of the English Language*, and discovered that its definitions took up three-quarters of page. I tentatively concluded that "run" must be the English word with the most definitions.

> Some authorities have since nominated the word "set" for that distinction, but I remain unconvinced. The controversy is further analyzed elsewhere.

Of course, both words are cases in my point here -- that English would surely be enriched if more lexical creativity were at play, by which heavily laden little words would be gratefully unburdened. By the way, I used "run" on the screen since the word "execute" would not fit in the limited space.

{Return to citation in Part 0}

Boolean Algebra: symbolic logic

Although various abbreviations were accomplished through symbols, even in the works of Aristotle himself, the use of symbols in a formal system, the precursor of modern symbolic logic, began with George Boole (1847) and Ernst Schröder (1890-1905). The system was developed further by Gottlob Frege (1879) and finally culminated in the *Principia Mathematica* of Bertrand Russell and Alfred North Whitehead (1910-13).

-- *Encyclopedia Britannca*

{Return to citation in Part 1}

Validity Not Necessarily Veracity: syllogism

in logic, a valid deductive argument having two premises and a conclusion. The traditional type is the categorical syllogism, in which both premises and the conclusion are simple declarative statements that are constructed using only three terms between them, each term appearing twice (as a subject and as a predicate): "All men are mortal; no gods are mortal; therefore no men are gods." The argument in such syllogisms is valid by virtue of the fact that it would not be possible to assert the premises and to deny the conclusion without contradicting oneself.

-- *Encyclopedia Britannca*

{Return to citation in Part 1}

synecdoche

noun [Latin, from Greek *synekdoche*, from. *syn-* + *ekdoche* sense, interpretation, from *ekdechesthai* to receive, understand; akin to Greek *dokein* to seem good]: a figure of speech by which a part is put for the whole (as fifty sail for fifty ships), the whole for a part (as society for high society), the species for the genus (as cutthroat for assassin), the genus for the species (as a creature for a man), or the name of the material for the thing made (as boards for stage).

-- *Merriam-Webster's Collegiate Dictionary*

In the seventies "Detroit iron" was the synecdoche used to pejorate vehicles manufactured by domestic automobile companies. "Silicon Valley" is a synecdoche, so is "dot-com." Likewise "hardware" for computer equipment -- the concrete things you can heft and haul.

{Return to citation in Part 1}

Retronym

The 1999 edition of the *Oxford English Dictionary* includes the following definition:

> **hardware** the physical components of a system or device as opposed to the procedures required for its operation; opp. software.

The earliest citations include...

> **1947** D. R. Hartree *Calculating Machines* "The ENIAC. I shall give a brief account of it, since it will make the later discussion more realistic if you have an idea of some ‘hardware’ and how it is used, and this is the equipment with which I am best acquainted."
>
> **1953** A. D. & K. H. V. Booth *Automatic Digital Calculators* "The engineering difficulties encountered in this type of machine are great, and a considerable increase in the size and complexity of the ‘hardware’ seems inevitable."

Accordingly, the word "hardware" was decidedly *not* a retronym, having made its attested appearance as applied to computing machines by 1947 -- a half-dozen years *before* the coinage of the word 'software.' That's six years by my reckoning -- not 13 years as depicted in the *OED* citations.

Readers are invited to observe the exclamation point at the end of the next sentence. In a rapidly developing realm, it seems doubtful indeed that such an obvious word relationship would have taken more than a dozen years to be recognized!

{Return to citation in Part 1}

Plug-Board Programming

A biographical sketch of one of the first computer programmers, Kay McNulty Mauchly Antonelli, appeared in the October 27, 2000 edition of Atlanta *Journal-Constitution*. She is the widow of John Mauchly who with Presper Eckert built the ENIAC in 1945. The passage most relevant to the present subject: "There was no such thing as software. ENIAC was programmed by wiring."

{Return to citation in Part 1}

Program Bugs

In about 1949, according to the mythology of the 1950s, a hapless moth flew into the Harvard Mark II. The technician taped the insect's corpse alongside the entry in his log "first actual case of a bug being found."

There was no verbal distinction in those days between a hardware failure and a programming error, both merely being deplored as "bugs." For some people the distinction is still pending (see Software Does Not Fail).

{Return to citation in Part 2}

Word: One Microsyllable

In "Digital Computers" (American Mathematical Monthly, Vol. 62, No. 6. Jun. - Jul., 1955, pp. 414-423), Mina Rees quoted the late Claude Shannon (1916-2001) as saying "a digital computer must be instructed in words of one microsyllable."

My friend Rich Alexander sent me Claude Shannon's obituary accompanied by this note:

> Regarding the first use of 'software,' you might be interested to know, I checked one of my all-time favorite books, *Symbols, Signals and Noise (*Harper & Row, 1961), written by J. R. Pierce, the top guy at Bell Labs and a buddy of Claude Shannon. The book focuses on information/communication theory, not computers, but there are passages about computers in which he uses these software-related terms: commands, compiler, instructions, program (noun), program (verb), programmer, and sequence of instructions. Pierce, the head of Bell Labs, does not use the word 'software.'

{Return to citation in Part 2}

Land Mark Studies at ITTE

Here is a partial list of the projects on which I worked during my two years (1953-1955) at the Institute for Transportation and Traffic Engineering.

- **Automobile Crash Injury Research** -- The world's earliest crash tests using real automobiles and anthropometric dummies to ascertain the efficacy of various restraining devices (lap belt, shoulder harness, chest-level strap) in preventing injuries to passengers resulting from frontal collisions; the research was cited prominently in Ralph Nader's landmark book *Unsafe At Any Speed*. A certain undergraduate authored the first published article on this research in the April 1954 edition of *California Engineer*, and the paper entitled Engineered Automobile Crashes re-appears on this website 50 years later by permission of the publisher.
- **Radar Speed Meter** -- The first such instrument being larger than a breadbox mounted on a tripod, with primitive klystron oscillator; controlled experiments confirmed the predicted accuracy and "cosine error in favor of the driver"; conducted successful demonstrations for law enforcement officials and legal experts from the eleven western states.
- **Padded Dashboard** -- Laboratory testing (accelerometry using an impacting pendulum) of various materials to determine the most effective, asymmetrical resiliencies for minimizing injuries from the "secondary" (inside-the-vehicle) collision; drafted recommendations to lawmakers resulting in Federal mandates for inclusion as standard equipment in new cars beginning in the sixties.
- **Vehicular Metering** -- Computer analyses ("Monte Carlo simulation") and experimental confirmation of various causes of traffic viscosity, including marginal friction and critical absorption volumes that cause "density lock" (now known by the misnomer "gridlock"), which adversely affects both the capacity and the mean transiting speed of a given highway system; the

work resulted in locally deployed metering lights, which are now widely applied to relieve congestion.

- **Cable Barrier** -- Analysis and testing of a UCLA-initiated system to assure opposing lane separation on expressways, designed to prevent both head-on intrusion hazards and rebounding damage resulting from post-crash control-loss; a system characterized by longitudinal cable supported by frangible aluminum posts, that were ultimately applied nationwide throughout the 1980s and into the 1990s.
- **Signage and Signal Standards** -- Experimental work in support of the earliest "human factors" research into traffic safety impacts attributable to destination signs and direction indicators, lane striping and signal lights; also conducted controlled experiments in a unique driving simulator with a wide-angle, cinerama-like screen to gauge effects on drivers subjected to fatigue and to various physiologically active substances.

There can be no doubt that the innovative work carried out by the research team at UCLA's ITTE during the 1950s was profoundly influential and will be viewed with respect by generations to come. It was an awsome privilege for an otherwise undistinguished verbal prankster to be included on that team.

{Return to citation in Part 2}

Marilyn Monroe and the API

My reputation was first established during the automobile crash injury research at ITTE. Derwyn Severy, the project leader, subjected himself to a low-speed rear-end collision experiment in a donated 1939 Plymouth, sort of in the tradition of Col. John Stapp and his contemporary rocket-sled tests. Remember those "news reels"? I was in charge of the high-speed motion picture photography.

Derwyn wore earplugs to prevent anticipation of the impact by the test car approaching from behind, and to assure that he would keep his eyes straight ahead, I affixed a purloined copy of the famous -- notorious -- Marilyn Monroe calendar upright on the hood. When the film was developed, we all gaped at the screen, astonished to see that, prior to the intended whiplash, the camera had faithfully recorded a hundred feet of Derwyn Severy grinning at a picture, which, though foreshortened, was all too immodestly discernible in each frame.

It was my turn to get a sore neck when the experiment was repeated for presentation in public forums. Without the calendar, of course.

At the conclusion of our first seatbelt study, I drafted the instrumentation chapter of the final report about the effectiveness of restraining devices on vehicular safety and prepared another section with a diagram describing one of my own early inventions. It was a micro-switch mounted under the cushion of a car-seat which was activated by the weight of a passenger and, in conjunction with the corresponding seatbelt switch, sounded a warning when the belt is left unfastened. In the diagram, I labeled it "API."

Nobody on the research team bothered to ask me what the letters stood for, doubtless assuming it was an arcane term-of-art. Finally, after publication of the first report, Dan Gerlough got suspicious.

"Ass-Presence Indicator," I explained cheerfully. Suffering *profondément consterné*, Derwyn Severy promptly issued a memorandum mandating that replies to all outside queries must define API as "Auxiliary Passenger Interlock." To the best of my recollection, no reviewer or journalist ever asked.

{Return to citation in Part 2}

Frivolicoin...

...might be offered herein as a self-referent addition to *Sniglets* by Rich Hall and Friends, Illustrated by Arnie Ten (The MacMillan Company, 1984), probably the most popular of several humorous collections of frivolous neologisms ("Any word that doesn't appear in the dictionary, but should").

In a separate work entitled *101 Words I Don't Use*, I have confessed to several coinages, including 'edutainment', 'circloid', 'holomorph', 'pluplural', 'patientoid', 'polycut', 'reprographics', 'totorial', and 'tridecabillion'. Not all of them were frivolous.

{Return to citation in Part 7}

Misnomer

The hardware in the SWAC was always breaking down. But not the software. Once I got a program checked out (the expression "debugged," a term credited to the Navy's Grace Hopper, was not in common use in 1953), then it would never break.

> Software never wears out, never decays, never gets weak. In terms of durability, then, software is not soft.

Words and phrases that start with ‘soft’ often imply attractive – even essential – attributes in some contexts. ‘Soft’ means offering little resistance; easily molded, cut, or worked; malleable: plastic; not hard; yielding readily to pressure or weight; not firm; smooth or fine to the touch; not harsh or coarse; bland, not irritating; low-toned, not loud or strident; subdued not glaring or overly brilliant; mild, gentle, and caressing; mild; balmy; yielding; easily touched; compassionate…

**soft+boil, soft+copy, soft+drink, soft+focus, soft+footed,
soft+goods, soft+hands, soft+key, soft+lighting, soft+liner,
soft+money, soft+pedal, soft+rock, soft+shell, soft+shoe,
soft+skin, soft+soap, soft+spoken, soft+spot, soft+style.**

Some of these, along with undesirable properties of softness that are implied from other realms, act together to make ‘software’ a **misnomer**. ‘Soft’ means out of condition; flabby; not sharply drawn or delineated; lenient, not stern; weak; unmanly or, for that matter, unwomanly; informal; simple; feeble; easy; diminished in value or importance…

**soft+back, soft+ball, soft+cover, soft+headed, soft+hearted,
soft+job, soft+news, soft+nosed, soft+rock, soft+science,
soft+target, soft+tissue, soft+touch, soft+witted, soft+y.**

Excuse the immodesty, but shortly after I graduated from UCLA in 1955, I *fore*saw the incomparable "hardness" of software. That rather controversial concept is celebrated in my iconoclastic screed entitled Software Does Not Fail. First drafted in the seventies, the piece was initially offered in various versions to any number of publications throughout the eighties. Finally I published it myself in *Sophisticated: The Magazine*. Here is an exerpt...

> Hardware is 'hard,' which is to say concrete, not abstract.
> Hardware can be made, in some sense harder, more reliable.
> Hardware can be insulated, hermetically sealed, ruggedized, bullet-proofed.
>
> Harder the better, presumably. But never hard enough. Sooner or later, hardware fails.
>
> Software, being abstract, is -- well, 'soft'.   No reason to make software hard, though. Truth be known: It is because of its softness that software does not fail.
>
> Nothing Else Is Software. There is nothing softer than software. Other things may be as abstract as software but hardly softer.

It has given me considerable pleasure that such a little polemic has over the years been cited in technical articles, popular texts, and system specifications. The title, which was once reproached as unsound, is now widely recited as a simple declarative (always with the "does not," never with a "doesn't").

Other Languages Don't Use 'Software'

That 'software' is a misnomer may be the reason for an apparent resistance to it in other languages.

Catalan: *programari*	Irish: *bogearraí*	Russian: *programmioye*
Dutch: *programmatuur*	Indonesian: *piranti*	Spanish: *programa*
Finnish: *ohjelmisto*	Norwegian: *programvare*	Swedish: *mjukvara*
French: *logiciel*	Persian: *narmafzar*	Vietnamese: *phan mem*

Little wonder, considering the tepid connotations in 'softwords' -- *softball, softcover, softener, softheaded, softhearted, softish, softling, softwood, softy*.

Meanwhile, you have the strength and perminence of 'hardwords' -- *hardback, hardball, hardbeam, hardcore, hardcover, harden, hardfisted, hardhead, hardhearted, hardihood, hardline, hardness, hardscrabble, hardstand, hardtop, hardwired, hardwood, hardworking, hardy*.

Some 30 venerable examples attest to the popularity of 'warewords' -- *agateware, barware, brassware, chinaware, clayware, cogware, cookware, copperware, crackleware, dinnerware, earthenware, enamelware, flatware, glassware, graniteware, greenware, henware, hollowware,* *honeyware, ironware, kitchenware, lacquerware, lusterware, metalware, redware, seaware, silverware, slipware, stemware, stoneware, tableware, tinware, tupperware, willowware, woodenware.*

The most colorful 'warewords' are all derived from 'software': *abandonware*, *adware*, *beerware*, *bloatware*, *brochureware*, *careware*, *crippleware*, *firmware*, *freeware*, *groupware*, *malware*, *middleware*, *nagware*, *netware*, *postcardware*, *shareware*, *shovelware*, *spyware*, *vaporware*, *wetware*. The following sentence has the exclamation point that belongs at the end of the previous sentence. For a misnomer, that's a fabulous foundation for creative coinage!

{Return to citation in Part 7}

Debugging and Degaussing

The computer I used for teaching maintenance was the prototype RW-300. The machine had a weakness that became well-known to me -- a thermal run-away in one of the germanium transistors on a small printed circuit board called the "C-register read amplifier." Do I have a memory or what! I did not bother to fix the thing. Instead, every few days the computer would halt, and I took advantage of the opportunity to guide my students through a trouble-shooting procedure that invariably wound up with pulling out the board, feeling the hot transistor, waving it around to cool it off and plugging it back in again. Problem solved.

That computer was also used by programmers for debugging process control systems. They were instructed to call me whenever the computer stopped. Soon a necktied figure would be seen running into the room, pulling out some printed circuit board, touching one of its components, waving it around, plugging it back in, and running back out again. My stratospheric reputation, though undeserved, was nevertheless thoroughly enjoyed, not to say exploited.

Our first magnetic tape machine came with a "degausser," literally a black box with nothing more than a switch on it. To restore a tape to its pristine, unmagnetized state, one simply place the reel on top of the box, turned it on and back off again. An oscillating magnetic field, which radiated from inside the box, did the work.

One day, when I was sure a couple of programmers were watching, I carried out a more elaborate procedure. I put a reel on the box and turned it on. I turned the reel first clockwise then counter-clockwise. Next, I slowly lifted the reel with both hands upward, higher and higher, then over my head, turning gracefully in a circle while stepping away in a solemn pirouette. Finally, I placed the tape on a nearby table and reached back and turned off the degausser. "Gradually shrinks the hysteresis loop," I muttered. "Eliminates glitches."

You surely know the rest of that story. The degaussing dance was faithfully performed for years thereafter by succeeding generations of programmers, and then, and then... Witnesses are plentiful who will testify that the following incident actually occurred:

Returning to the California offices of TRW from a year of research on air traffic control in New Jersey, I strolled into the computer laboratory. During my absence, the C-register read amplifier on the prototype RW-300 had behaved itself perfectly. A few minutes after my arrival, though, the computer stopped. Tah-dah. A new batch of programmers were waiting to be astonished. I could not let them down. Out of such coincidences, gods are created.

{Return to citation in Part 7}

AIEE + IRE = IEEE

American Institute of Electrical and Electronic Engineers (AIEE) and Institute of Radio Engineers (IRE) merged to become the Institnte of Electrical and Electronic Engineers (IEEE, pronounced I-triple-E).

{Return to citation in Part 7}

Seven Dwarfs

There must have been more than seven, come to think of it, since any list at about that time would surely have included Alwac, Bendix, Burroughs, Control Data, Data General, Digital Equipment Corporation, General Electric, Honeywell, National Cash Register, Radio Corporation of America, Remington Rand Univac, Scientific Data Systems, all aspiring to challenge IBM in computers -- each, as we used to say, "like a lecherous mouse climbing the hindleg of an elephant with a smirk on his face."

{Return to citation in Part 7}

Mutually Dependent Industries

In *American Mathematical Monthly* (Vol. 72, No. 2, Part 2: Computers and Computing. Feb., 1965, pp. 8-14), R. D. Richtmyer wrote about the emerging utilities that were necessarily being provided by hardware purveyors, invoking thereby a narrow, not to say parochial definition of 'software.'

> The compiler is usually embedded in an elaborate system for using the computer, which sequences the problems through the computer, one after another, and supervises them in various ways. This system, together with the library of subroutines, routines for control of input and output formats, and so on, has come to be called software in the industry (as opposed to hardware); a manufacturer is expected to supply a considerable amount of software with the machines he sells.
>
> A curious phenomenon that has accompanied the development of software is a tendency for the hardware to become dependent on it. The amount and complexity of the software have increased enormously in the last few years, and its preparation has become a rather large industry.

Thus, from the perspective of a mathematician in the middle '60s, computer hardware depended on a class of programming -- language processing and operating systems that provided support services for the programmer and called that class of non-application programming 'software,' whereas in a broader context, computer hardware had from the earliest days been called upon to execute software-intensive applications -- science, business, military -- without the benefit of that class of non-application programming.

{Return to citation in Part 7}

Thicket in the Bilge

Throughout the seventies, Xerox was perceived as one of the two best run enterprises on the planet, the other being IBM. One could not help wondering if the mere mortals running the company read those rhapsodic pieces each day about themselves in *The Wallstreet Journal* and *Forbes*, *Barrons* and *Business Week* and asked, "Who me?" At corporate headquarters of The Grand Old Duplicator Company, I can tell you, self-doubt was nowhere in evidence, and in its place there was plenty of unmerited self-approval.

Perhaps I might have been more discrete, but to my close friends I described Xerox top managers as...

> grim-faced officers standing in the bridge of a venerable vessel, steadily plowing the commercial ocean toward some unseen Valhalla in the distance, each in his sharply pressed uniform ornamented with gleaming brass and rows of ribbons, their caps and collars festooned with scrambled eggs, all eyes gazing at the horizon, hands confidently operating levers and wheels, each unaware that the controls were *not connected to anything*, that instead the equipment in the bowels of the ship had been welded firmly in place, and that a unique technological marvel -- a xerographic engine, protected by "the thicket of patents" -- would assure more than enough financial propulsion to conceal their own buffoonery.

What good are metaphors if you don't mix them?

{Return to citation in Part 7}

The MANIAC

The first giant brain equipped with a Teletype was the MANIAC. In a secret demonstration for a group of high-level Pentagon officials, a four-star general was invited to ask the machine a question.

With two fingers, the general pecked the keys solemnly, "WILL THERE BE A THIRD WORLD WAR?"

For a full minute, the MANIAC made whirring sounds, its panel lights flickering, then typed one word, "YES."

The general and his entourage gasped. He leaned over the keyboard, frowning. "YES, WHAT?" he pounded.

Taking longer this time, the MANIAC made more whirring sounds, its panel lights flickering wildly, then typed, "YES, SIR!"

{Return to citation in Part 7}

Lost Satirical Cartoons

Lost from my collection of satirical cartoons is one from 1954 in which the scene was a destroyed city. Smoldering ruins are strewn as far as the eye can see. Rubble and broken bricks are spread out in the foreground alongside smashed cars and toppled utility poles. Twisted steel girders are all that remains of a tall building, with a bathtub and a toilet held aloft by their plumbing. A public address speaker can be seen dangling by its wires. The caption reads, "All clear."

{Return to citation in Part 5}

Another is one from 1974 featuring a row of vending machines. A sign in the background reads "Baggage Claim," establishing the setting as a transportation terminal. A terse placard appears on one of the machines: "CHANGE." On the floor in the foreground atop an open paint can droops a brush dripping red paint. Defacing the machine in crude, red lettering is a forlorn battle-cry, "CONTINUITY!"

{Return to citation in Part 6}

<< **Previous**	**0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**	**Next Part** >>
Home Page	**Softword: Table of Contents**	**Top of Article**

metaphor

Software Does Not Fail

Paul Niquette

Updated in 2006 for inclusion in
Softword: Provenance for the Word Software

Part 1
The Meaning of Work

Do not read the title again. I'll repeat it for you: Software does not fail.

"Everything fails," people say. "Hey, especially software."

- People who say that include journalists, of course. They enjoy reporting the adversities people have suffered since the beginning of the 'software age.'
- Then, too, people who have more than a little knowledge of software -- software experts, even -- say that software fails.
- Writers of software engineering textbooks say that software fails.
- Universities employ lecturers in mathematics and logic design, in systems engineering and software engineering all telling their students that software fails.
- Decision makers at every level of management in industry and every department of government say that software fails.{*HyperNote 1*}

Whatever their intelligence, their knowledge, their experience, these people have one thing in common.

> They are not sophisticated.{*HyperNote 2*}

Software either works or it does not work.

> To be said to work, software must do what it is supposed to do. A specification determines what software is supposed to do -- and by implication what software is **not** supposed to do.{*HyperNote 3*}

Software that does not work has not failed. It did not work in the first place.

> To be said to fail, software would have to work in the first place then not work in the second place. Software does not do that.{*HypeNote 4*}

Neither Moth nor Dust

If software ever worked, it will always work. Always.

- Software does not wear out.
- Software does not decay.
- Software is not vulnerable to environmental insults.
- Temperature, hot or cold, has no effect on software.
- Software needs no protection from shock and vibration.
- Neither moth nor dust doth corrupt software.

People think that software worked in the first place and then stopped working.

> They are not sophisticated.

Hardware Fails

To work, software must run, which means commanding hardware to do what hardware is supposed to do. Software, then, depends upon hardware. And hardware...

- does wear out,
- does decay,
- does suffer from temperature extremes,
- does need protection from shock and vibration,
- does get eaten by moths and clogged up by dust.

Hardware Always Fails

It is possible for hardware to work and then to stop working.

> It is more than possible, it is certain.

No question about it, when hardware works and then stops working, that's a failure.

Repair Applies to Failures

A hardware failure can be fixed by replacement of the hardware in whole or in part -- replacement by **identical** hardware, in fact. That's what 'repair' is.

> Since software does not fail, 'repair' has no meaning.

If software does not comply with the specification, replacing it in whole or in part with identical software will not make it comply with the specification. Replacing software that does not work with **redesigned** software that does work is not 'repair.' {*HyperNote 5*}

Reliability Applies to Failures

Reliability is a term which has meaning only for hardware not software. Reliability is defined as the probability of **not failing** within a given period of time.

Software does not fail within **any** period of time. The statistical expression 'mean time between failures' (MTBF) has no meaning for software, inasmuch as the inverse, 'failure rate,' is **zero** for software.{*HyperNote 6*}

Concrete Versus Abstract

Hardware is 'hard,' which is to say concrete, not abstract. Hardware can be made, in some sense harder, more reliable. Hardware can be...

- insulated,
- hermetically sealed,
- ruggedized,
- bullet-proofed.

Harder the better, presumably. But never hard enough. Sooner or later, hardware fails.

> By the way, humans are hardware.

Software, being abstract, is -- well, 'soft.' No reason to make software hard, though.

> Truth be known: It is because of its softness that software does not fail.

Nothing Else Is Software

There is nothing softer than software. Other things may be as abstract as software but hardly softer.

> A 'procedure,' for example, is abstract and therefore as soft as software.

Software is a procedure but not all procedures are software. In fact no other procedures are software.

> Software runs on hardware; other procedures run on a special kind of hardware: humans.

People like to call books and films and videos "software."

> They are not sophisticated.

The Specification

The specification for either hardware or software is abstract and therefore soft. The specification does not do what software does, however. So the specification cannot be said to work or not work.

The specification, which is derived from functional requirements, can be dichotomized many ways:

- right or wrong,
- complete or incomplete,
- appropriate or inappropriate,
- useful or useless,
- safe or dangerous.

To go from one to another, the specification itself would need to be 'revised.' Or the functional requirements would need to be 'amended.'

One Dichotomy

Only one dichotomy applies to software: either it works or it does not work, which is determined by whether the software complies with the specification. The same can be said for hardware; however, a failure takes hardware from working to not working; conversely, repair takes hardware from not working to working.

- To go from working to not working, software would have to be **redesigned**. There would be no reason to do that, of course.
- To go from not working to working, software would have to be **redesigned**. There is a reason to do that.

For the record, then, complying with a specification that is...

- wrong,
- incomplete,
- inappropriate,
- useless, or
- dangerous

...does not mean failure of either hardware or software.{*HyperNote 8*}

Objects

What software does is access 'objects' and assign values to 'objects.'

- Objects can be sets of variables or constants.
- Objects can be inputs from hardware or from software.
- Objects can be outputs to hardware or to software.
- Objects can reside in memory, which is hardware. {*HyperNote 9*}

Accessing an object in memory is called 'reading.' Assigning a value to an object in memory is called 'writing.' Software does something else, too.

Internal States

Software references its 'internal state' and changes its 'internal state.'

- Accessing objects in memory is how software references its internal state.
- Assigning values to objects in memory is how software changes its internal state.

Hardware does what software does, but hardware does something else, too. Hardware fails.

Cases

To be said to work, software must comply with the specification in all 'cases.' Not some, all.

A 'case' is merely a combination of input values.

Now, depending on its 'internal state,' software generally does something different with a given combination of input values. A given case, then, is not sufficient to determine what software does.

Sequences

Each internal state is reached as the result of software being acted upon by a sequence of input values -- cases. Software changes its internal state whenever necessary to comply with the specification. Thus, a sequence of cases is necessary to determine what software does.

> A sequence of cases will be necessary but not sufficient to determine what software does -- unless the sequence begins with the software in a known state such as 'reset' and the sequence is long enough.{*HyperNote 10*}

The statement that software must be designed to work in all sequences of cases is general enough to accommodate software's ability to reference and change its internal state.

Doing the Right Thing

Software necessarily does the same thing in each sequence of cases -- the right thing. If what the software does in a given sequence of cases complies with the specification, the software is said to work -- for that sequence of cases.

> Not doing what it is supposed to do in a given sequence of cases does not mean the software has failed. Instead, it means software did not work in the first place.

Not only that but the specification must determine software's response to a class of inputs called 'interrupts,' which are distinguished by the fact that an 'interrupt' can directly change software's internal state -- but not without the authorization by software. The specification must also determine software's treatment of the 'non-maskable interrupt' and 'reset.'

> Nothing can be left to chance, here.

Specification in Code

To do what it does, software must run on hardware, which means commanding the hardware on which it runs.

> Commands are codes, and 'code' results from a procedure called **design**, which is what 'software engineers' do.

The specification is not code. More than one code may comply with the specification. Each code can be designed differently and produce a different sequence of internal states.

> A given code can be different from another code that works and still work.

In other words, a specification determines what software is supposed to do but not **how** the software is supposed to do it. A specification must only be explicit enough to determine what values software will assign to output objects in each sequence of cases -- the external consequences of each sequence.

But a specification does not determine the sequence of internal states. If a specification did that, the specification would be more than a specification. It would be code.{HyperNote 12}

A sophisticated person would describe 'design' as a process not a procedure, the latter implying the power to determine. Code is not determined by a procedure.

Design is a rigorous process guided by knowledge.{HyperNote 11}

Software Does Not Fail

Part 2
Quality Assurance

The expression 'quality assurance' applies to software. The term also applies to hardware.

Quality assurance differs from 'quality control,' which applies only to hardware -- specifically to hardware manufacturing. Being abstract, software is not, in the conventional sense, manufactured.

Quality assurance means proving that a given code works. Which may not be easy. It is worth the effort, though. Once proven to work in all sequences of cases, software does not fail.

To be said to work, quality assurance must prove that software works. More to the point, quality assurance must not allow software that does not work to be said to work.

The previous sentence deserves to be read again.

Quality assurance is a procedure and therefore soft -- as soft as software. Quality assurance either works or it does not work.

Quality assurance is not a matter of degree. It is all or none. And 'all' can be plenty.

The magnitude of quality assurance depends on the complexity of software. Complexity is indeed a matter of degree. Degrees, plural.

- Complexity means size, for one thing: number of lines of code. Each line of code applies to some number of cases and sequences of cases for which the software must be proven to work.
- Complexity increases with the number of input objects and the range of values each variable can take on.
- Complexity increases with the number of possible internal states, too.
- Complexity increases with the number of sequences in which those states can occur.

The quantity of cases and sequences of cases can be exceedingly large but not infinite. Quality can in principle always be assured.

Testing: Myth and Reality

Testing is the best way to assure quality, some people say. They are not sophisticated.

> Whereas proving that software works in a given sequence of cases can indeed be accomplished by testing, testing one sequence may or may not prove that the software works in another sequence.

If the software does not do what it is supposed to do in a given test, the software does not work. On the other hand, if the software does do what it is supposed to do in a given test, the software **may or may not** work.

> Quality assurance requires all possible tests. That may not be practical.

A typical 'test' would be prepared as a sequence of input objects that are systematically caused to act upon the software. The sequence of values assigned by the software to the output objects would then be observed. The output sequence, having been independently predicted, would need to be compared with what the software actually outputs.

> Tests generally come in great numbers, but often tests do relate to each other.

Cleverness: The Myth

A clever selection of tests might make quality assurance by testing practical.

> In the previous sentence, 'might' appears in the predicate. That's because 'clever' appears in the nominative.

The clever **selection** of a test from all possible tests can be exceedingly difficult -- more difficult, in fact, than designing the code in the first place. That testing is not the best way to assure quality can be demonstrated for software of any complexity.

> Even simple software.

Simple Example

Consider the following simple specification:

1. Software is required to assign a value to one output object based on the value of one input object. The output object is a bit, and the input object is a byte.{*HyperNote 13*}
2. Starting out in a 'reset' state, the output is set to a value of **zero**.
3. Whenever the binary value of the input is **greater than one hundred**, the output is set to the value **one**, which is sustained until...
4. The binary value of the input equals **zero**, whereupon the output is assigned a value of **zero**.

Many practical applications rely on such a specification. The input object may represent...

- pressure in a vessel or
- traffic on a communications network or

- radiation from a reactor or
- strength of a wireless signal or
- speed of a vehicle.

...the output object in each case being used to allow or to limit some action.

> Software does not get much simpler, and testing that software would not be exceedingly difficult.

Trivial Sequence

There are merely two 'internal states' required to fulfill the specification in the Simple Example: Call them 'on' and 'off,' which, in conjunction with the input object, determine the value of the output object to be one and zero.

- Starting out in the 'off' state following a reset, an input with value zero must produce an output with value zero. If the software outputs a value one, the test will have proven that the software does not work. A zero on the output does not prove that the software works.
- Same for an input value of one, two, three, on up to 100 -- all with an output value of zero.

After 101 tests, then, the software would **not** have been proven to work. The software may, in fact, be utterly ignoring the input values.

- At least one more test is needed -- a test having an input value **greater than 100.** If the output value is still zero, that test will have proven that the software does **not** work. Nevertheless, observing a value of one on the output does **not** prove that the software works (only that the software is not utterly ignoring the input values).

After 102 tests, then, the software has **not** been proven to work. There are still 154 more tests to go. {*HyperNote 14*}

- Once the internal state switches from 'off' to 'on,' the output remains at a value of one for all values of the input except zero, whereupon the software changes its internal state from 'on' to 'off.'

Cleverness: The Reality

For the Simple Example, tests do indeed relate to each other. There are four sets of cases*:*

1. those which are specified to leave the internal state 'off,'
2. those which are specified to switch the internal state from 'off' to 'on,'
3. those which are specified to leave the internal state 'on,'
4. those which are specified to switch the internal state from 'on' to 'off'

For quality-assurance-by-testing, sets numbers 1 and 2 are the most interesting, and there would be 256 cases to be tested. It is tempting to suppose that a 'clever' selection of cases will make quality-assurance-by-testing practical. There seems to be no need to bother with input values much above or below the dividing line, 100. The idea would be to concentrate testing in the range of, say, 90 to 110.

That will **not** work. Here's why.

The Rational Software Engineer

Assume there to be a rational software engineer who, given the specification for the Simple Example, would yawn and then design his or her software to perform a simple arithmetic comparison between the binary value of the input byte and a constant value of 100.

A rational software engineer can make a mistake in the coding, though.

For example, he or she might have neglected to code the software to treat the input object as an **unsigned integer**. If so, then for all tests of input values higher than 127, the software would interpret the binary number as **negative**.{*HyperNote 15*}

- Thus, for input values from zero through 127, the software would do what it is supposed to do, leaving the internal state 'off' for inputs from zero through 100 then switching the internal state 'on' for inputs above 100.
- Surprise: For all input values from 128 through 255, the software does not do what it is supposed to do: Instead the software leaves the internal state 'off,' and the output stays at zero.

Accordingly, the software does not work.

Finding Out the Hard Way

For that particular mistake in code, then, a test having an input value larger than 127 would prove that the software does not work. On the other hand, quality assurance based only on a 'clever' selection of tests near the value 100 would allow software with a design mistake to be said to work.

Then one of the untested input values comes along and poof:

- the vessel explodes or
- the network saturates or
- the reactor melts down or
- the radio signal gets lost or
- the vehicle crashes.{*HyperNote 16*}

Hardly the preferred way to find out that the software -- what? -- did not work.

Quality Assurance Did Not Fail

Some people will say it was the quality assurance that failed.

They are not sophisticated.

Quality assurance does not fail. For the Simple Example, what was demonstrated is...

- that testing with input values less than 128 did not assure quality of the software and
- that a test of merely one input value larger than 127 would have worked

...by proving that the software did **not** work.{*HyperNote 17*}

It is not appropriate to say that quality assurance worked for a few tests **and then stop working**. That would be the evidence of failure. Instead, quality assurance did not work, period.

Cleverness: The Impossibility

If performing all possible tests is not practical, then quality-assurance-by-testing will work if and only if the selected tests are the ones that do not allow software that does not work to be said to work.

> The 'clever' selection of tests to assure quality can be exceedingly difficult.

A sophisticated person will say that it would be impossible to select tests that assure quality **without knowledge of the code**. Some people will say that 'improbable' is a better word, but software deserves more stringent treatment:

> Quality assurance is not a matter of probability.

Knowing the specification is not enough, as we shall see.

The Reasonable Software Engineer

Consider the same Simple Example that was coded by the rational software engineer in the hands of a different person -- a reasonable software engineer. After yawning, he or she designs the software to use a **table look-up** procedure.

Rather than arithmetically comparing the binary value of each input byte to a constant 100, the software is coded to treat the byte as a 'pointer' which references an object in memory called a 'bit-map.'

- The first 101 entries are zeros, the rest are ones.
- The zeros are used to leave the internal state 'off,' the ones are used to switch the internal state from 'off' to 'on'.

There are reasons for the reasonable software engineer to adopt this more general design approach. A 'translator,' for example, would have called for a byte as the output, which is best coded as a table-lookup.

> Note that the code is different, but the specification is the same.

Let us again postulate that quality assurance is to be achieved by testing -- **without knowledge of the code, only the specification**. As before, there are 256 possible tests. The problem would be to select fewer than all possible tests and still assure quality.

- The reasonable software engineer might make the same coding mistake as the rational software engineer.

If so, for tests of input values greater than 127, the pointer would point outside the table and there's no telling what the value of the output bit might be -- including the correct value, of course. During testing. Later, after a revision to the software, tah-dum, those out-of-bounds references -- to objects having nothing whatever to do with the software segment at hand -- will contain different and therefore incorrect values. It happens.

- The reasonable software engineer might make a different mistake, though, like putting a one in the table where a zero should be or vice versa.

There are 256 ways to make one such mistake. Any of them will make the software not work.

Quality-assurance-by-testing would not work unless the selected tests happen -- by pure chance -- to include the one or more in which the software does not work. After the software is put into use and the untested input value arises, the consequences would be attributed to a software failure by some people -- and to quality assurance failure by other people.

A sophisticated person would say...

- that through a mistake in the design of the code, the software did not work, **and**
- that through what might be called 'bad luck' in testing, quality assurance did not work.

The tests that might have proven that the software does not work in the first place were not performed, so the software was said to work.{*HyperNote 18*}

The Simple Example is not uncommon and the coding errors postulated above are not far-fetched. The 'software age' is with us. What ever shall we do? {*HyperNote 19*}

Software Does Not Fail

Part 3 The Code Walk-Through

Some will say that testing does not assure quality unless all possible tests are performed.

> Maybe that's what people mean when they say, "You cannot test quality **into** something."

In software, complexity makes testing all possible cases impractical, and making a 'clever' selection of tests that will assure quality without knowledge of the code is impossible. The next sentence may need to be read more than once.

> With enough knowledge of the code, testing is not necessary.{*HyperNote 20*}

First Principles Apply

Setting aside quality-assurance-by-testing, a sophisticated review of First Principles would include...

1. Assuring that a given thing will work requires knowledge of the specification for that thing plus...
2. Assuring that a given thing will work requires knowledge of how the thing works.
3. Knowledge of anything requires study.
4. Knowledge of how things work requires a special kind of study, called the 'Design Review.'
5. There needs to be a procedure for conducting the Design Review.
6. The procedure used for conducting the Design Review of software is called the 'Code Walk-Through.'
7. As a procedure, the Code Walk-Through is abstract and therefore soft.
8. Being abstract and soft means that the Code Walk-Through either works or it does not work.
9. To be said to work, the Code Walk-Through must find all mistakes in the code. Not some, all.
10. The Code Walk-Through runs on a special kind of hardware: humans.
11. The Code Walk-Through does **not have the power to command the hardware** on which it runs.
12. The Code Walk-Through, therefore, is not software; nevertheless,...
13. The Code Walk-Through does not fail. Instead,..
14. The Code Walk-Through which does not find all mistakes did not work.

Undersight

The human conducting the code walk-through can make an 'undersight,' of course, much as a software engineer designing software can make a mistake. {*HyperNote 21*}

- An undersight during the Code Walk-Through will result in undiscovered mistakes in the code.
- A different human conducting the same Code Walk-Through may not make an undersight.
- For that matter, the same human repeating the Code Walk-Through may not make an undersight.

If there is no undersight but not all the mistakes are found, the Code Walk-Through did not work.

- Performing the same Code Walk-Through will not make it work.
- The Code Walk-Through, which is a procedure, needs to be revised to make it work.

Unlike quality-assurance-by-testing, the Code Walk-Through cannot be automated much.

- Compilers identify a few classes of mistakes. Other tools are less effective.
- Automatic monitoring for 'code coverage,' for example, is utterly inconclusive.

Unlike quality-assurance-by-testing, the Code Walk-Through is mind labor. Like designing software.

Soft Things Made Soft

Quality assurance by the Code Walk-Through can be exceedingly difficult. But no more difficult than designing the software itself. And far less difficult than testing all possible sequences of cases.

Consider the Simple Example again. Only a few lines of code were needed to meet the specification.

- The rational software engineer made a mistake in one of those lines: assigning the wrong 'type' to a variable. The Code Walk-Through that checks to make sure all variables have been 'typed' correctly would have found the mistake -- assuming the human performing the code walk-through does not make an undersight.
- The reasonable software engineer made a mistake in coding the bit-map. A code walk-through that includes the inspection of all entries in tables would have found that mistake -- again, assuming the human does not make an undersight.

Performing the Code Walk-Through of a few lines would surely be less difficult than quality-assurance-by testing, which means...

1. preparing a complete set of values as input objects,
2. causing each to be accessed by the software,
3. independently predicting the values that are to be assigned by the software to the output, and
4. comparing the predicted outputs with those observed in the consequent outputs.

Then too, many of these steps are vulnerable to human error -- undersights.

Managing Complexity

Some people will say that complexity makes the Code Walk-Through not practical.

> They are not sophisticated.

If the code is too complex to walk-through, it is too complex to design -- and too complex to test, since without knowledge of the code, quality assurance requires testing all sequences of cases, which is impractical.

> The sophisticated person knows that complexity must be managed.

Managing Practicality

One way to do that is to partition the code into 'modules.' A specification must then be prepared for what each module is supposed to do and for what all the modules together are supposed to do.

> Unfortunately, quality-assurance-by-testing requires preparation of sequences of cases for each module and some method to intercept the consequences -- 'stubs.' All software modules need to be tested together, which means that testing all possible sequences of cases will be impractical, even with modular software.

On the other hand, if the code is modular, the Code Walk-Through can indeed more readily assure quality of software.

The practical becomes more practical.

The Sophisticated Code Walk-Through

The sophisticated Code Walk-Through must include procedures...

- that perform detailed analysis of branch topology and case statements,
- that evaluate the effect of processor loading on realtime performance,
- that calculate demands and limitations of data rates and processing delays,
- that validate each flag and pointer in function calls,
- that postulate hardware failures to ascertain remedies and responses,
- that review all interrupt processing for hygienic and timely context switching,
- that analyze asynchronous events for vulnerabilities,
- that examine each loop for accuracy in count and closure,
- that verify the initialization of each iterative sequence,
- that review internal and hoisted processing of loop parameters,
- that check for inclusions and exclusions in code topology,
- that estimate loop timing,
- that validate re-entrant code,
- that review each coded logic decision for correctness,
- that audit the allocation and ranging of heap resources,
- that review the stack management algorithm for proper function,
- that authenticate matrix design for hierarchical consistency,
- that make sure all n-tuples are well-formed and all don't-care cases are properly terminated,
- that check the base address and offset range for pointers,
- that study software partitioning for matched interfaces between modules and coordinate processing,
- that check for violations of enterprise-specific policies,
- that assure compliance with protocols and absolute correspondence between requests and services,
- that check typical and extreme values in recursion formulas,
- that audit for adherence to software design practices of professional societies and standards organizations,
- that confirm all entrances and exits conform to structured programming conventions,
- that inspect table entries for accuracy in coding,
- that inspect all variable attributes for conformity to specification,
- that assure all variables have been typed and scoped correctly, and
- that inspect to make sure that the code has been annotated clearly and completely -- hey, and correctly.

A human must perform all of those procedures -- and more -- with no undersights.The consequent software may still not work.

Whatever the ultimate result, though, software does not fail.

HyperNotes

{1} **References That Fail**

See for example Pressman, Roger S., *Software Engineering* (1987, McGraw-Hill) or von Mayrhauser, Anneliese, *Software Engineering*(1990, Harcourt Brace Jovanovich). As for hearsay references, suffice it to say that forcible expressions of these sentiments have reached my ears from a hundred sources since the mid-fifties. {*Return*}

{2} **Cannon-Balls**

My license for bluntness here is appropriated from Ralph Waldo Emerson (1803-1882).

> "Speak what you think in words as hard as cannon-balls..." {*Return*}

{3} **Unintended Features**

Permit me to quote William Makepeace Thackeray (1811-1863):

> "For the wicked are wicked indeed and they go astray and they fall and they come to their just desserts, but who can tell the mischief that the very virtuous do!"

Words well suited to lamenting the unintended consequences of features gratuitously designed into software. {*Return*}

{4} **Give me a break.**

Some people will say that not working in the first place is just another expression for 'failure.'

> They are not sophisticated.

An immense distinction is being presented here. We need to restore the verb 'fail' to its full meaning. One might substitute 'break'-- but only for hardware. Nobody says software breaks. {*Return*}

{5} **Words to Work By**

Terms selected here are somewhat arbitrary but intended to serve clarity by virtue of consistency, thus one might...

> 'amend' requirements,
> 'revise' specifications,
> 'perform' tests,
> 'repair' hardware,
> 'redesign' software.

Long ago somebody, who was obviously not sophisticated, appropriated the term 'maintain' for software. Software 'maintenance' in every meaningful respect...

> what and who,
> how and why

...does not even faintly resemble hardware maintenance. {*Return*}

{6} **References That Crash**

Unhelpful indeed are statements such as are found in Siewiorek, Daniel P. and Swarz, Robert S., *The Theory and Practice of Reliable System Design* (1982, Digital Press, Bedford, MA):

> "A failure causing a [computer] crash may be the result of either hardware or software failure."

The passage goes on to attribute software 'failures' to "the introduction of new features" or "previously undetected faults." {*Return*}

{7} **Softening Hardware**

Ironically, hardware gets effectively **harder** when more of its functions are given over to software, as in embedded controllers or, in the extreme, the 'reduced instruction set computer' (RISC).

> The term 'firmware' means software that resides in read-only-memory. *Nota bene*, firmware is as 'soft' as software. {*Return*}

{8} **Words in Use**

The distinction being drawn is between 'software' and 'hardware,' the latter can be taken to mean a 'computer,' 'processor,' 'controller,' whatever.

> Other words have been eliminated altogether as not relevant to the present scope 'program' and 'subprogram,' 'routine' and 'subroutine.' The word 'system' long ago got stretched beyond its elastic limits and lost its strength.

Professional 'programmers' were replaced by 'software engineers' in the mid-seventies; the other kind became 'hackers.' As a term of disparagement, 'hacker' has two meanings: 'cyber-bumblers' and 'cyber-burglers.' {*Return*}

{9} **Generosity to Generality**

The sophisticated term 'object,' which is general in the extreme, gives renewed evidence of a linguistic struggle dating back to the dawn of the 'software age' and the realization that calling 'software engineering' 'computer science' is tantamount to calling 'mechanical engineering' 'automobile science.' {*Return*}

{10} **Who cares?**

Not all cases can arise in real life, thus we have the term 'don't-care' for those cases.

The sophisticated person cares about those cases: first making sure that they cannot indeed arise and second making sure that if they do arise, the software works anyway. {*Return*}

{11} **Reference to Knowledge**

In *Cultural Literacy* (Random House, 1988), E. D. Hirsch frames a compelling picture of the knowledge-bound character of all cognitive skills. {*Return*}

{12} **Code of Honor**

The term 'code' is taken here to mean 'source code,' which is what software engineers produce.

Development software -- compiler or assembler -- produces 'object code', which is what actually commands hardware. The distinction is not germane to the present subject.

> Unless the development software does not work. {*Return*}

{13} **Taking a Byte**

The word 'byte' may be the purest neologism of the 'software age,' giving evidence of the forlorn and tardy impact of 'computer science' on the language we speak. I dare to use the term here without explanation. {*Return*}

{14} **Numbers to Count On**

The 256 test cases in the Simple Example are not too many to do, even by hand. A minor tweak of the example, though, to 16 bits on the input will change that.

> Automating the testing with a 'driver' means, for the Simple Example, writing more code to do the testing than the code in the first place. Another tweak or two, and the number of test cases -- even for the Simple Example -- will exceed the number of subatomic particles in the Milky Way Galaxy.

The issue resides, of course, in Sir Karl Popper's requirement for 'falsifiability'. {*Return*}

{15} Song of a Bit

The leftmost digit of a binary integer of a given length can be treated either as a binary bit or as the sign of a binary integer one bit shorter in length.

> In byte-land, the former interpretation will support a range of values from 0 through +255, the latter, values from -128 through +127. {*Return*}

{16} Reference to Nonsense

A compendium of the most sensational consequences of -- and misguided attributions to -- software failures appears in Leonard Lee's *The Day the Phones Stopped* (1991 Donald T. Fine, Inc), which inspired the present polemic. The book is listed by the Library of Congress under the category 'Computer software -- Reliability.' {*Return*}

{17} Bugs and Glitches

Most people like to say that if software works in some cases but does not work in other cases, the software has a 'bug' in it. That may be more meaningful than saying that the software failed. But not by a lot. The software did not **get** a bug in it. The software engineer made a **mistake.**

> A software engineer bemoaning a bug in his or her software is tantamount to a tennis player lamenting flaws in his or her own backhand -- hey, or concentration.

A few people like to say that if software works in some cases but does not work in other cases, the software has a 'glitch' in it. That is less meaningful than saying that the software failed. By a lot.

> When 'glitch' gets into a sophisticated dictionary, it will be given a synonym, 'intermittent.' {*Return*}

{18} Crap Dusting

One handy software engineering tool is the 'debugger,' which is a misnomer tantamount to calling a 'magnifying glass' an 'insecticide.' {*Return*}

{19} Mole Hill Mountain

The Simple Example appeared as a portion of a real-life specification, which called for the software to use the binary value of the input byte to select an object from memory for reading.

- The software engineer designed the software to treat the input byte as a 'pointer' to objects in memory.

- Making matters worse, each object was the beginning of a different procedure within the software.
- All possible values of the pointer were properly accounted for in the code.
- The pointer itself, however, was unintentionally coded as a signed integer and then treated as a conventional 'offset' added to a constant to form a memory address.
- The software passed quality assurance, having worked for all test cases and released.
- The software appeared to work at installations all over the world.

And then, and then... {*Return*}

{20}**Verified Validity**

However inconclusive, testing of software will continue to be practiced until the sun flickers from the sky.

> Software 'bureaucracies' brim with testing. And what is known as SVVP (Software Validation and Verification Plan, abbreviated "V&V"), an orthodoxy wherein both Vs are mandated: The first assuring that the specification is correct and the second testing to make sure the software does what the specification mandates -- in other words, works. {*Return*}

{21} **Oversight and Undersight**

The word 'oversight' like 'cleave' can cut both ways ("split apart" or "cling together").

> As "watchful care," the Code Walk- Through is 'oversight' at its best.
> As an "unintended omission," even one 'oversight' in the Code Walk-Through means disaster.

A sophisticated writer might use 'undersight' to denote the latter. {*Return*}

Home Page | Back to References | Top of Article

.