LEN SHUSTEK

July-September 1993 Volume 1.1

The Analytical Engine

NEWSLETTER OF THE COMPUTER HISTORY ASSOCIATION OF CALIFORNIA

EDITORIAL: Welcome to the Analytical Engine

....volume one, number one.

This document has three purposes: To present a small sample of computer history. To convince you that computer history is worth exploring and preserving. To persuade you that a modest commitment of your time or money, or both, will help build an institution and protect some of the most important scientific information in the world today.

Let me give you one concise example. Remember the Pong machine? It wasn't much to look at; a black-and-white CRT in a squat, screaming-yellow plywood box, with a couple of black knobs. But in the early Seventies it introduced thousands of people — and, beyond them, the world — to the interactive video game. It was one of the first computer-driven devices to become a memorable part of our culture.

Of the perhaps 15,000 Pong machines that Atari built, *there are fewer than a dozen left* and they are substantially priceless. Collectors compete to buy them. What happened to the others? They were displaced, replaced, thrown away. Junked.

Ever since the days of ENIAC, the rallying cry of computing has been *Let's chuck the old stuff to make room for the new stuff!!* and hardware is scrapped, with the hardware goes the documentation, goes the information, leaving only the thin thread of memory which snaps too. Leaving aside dubious precursors, electronic computing is fifty-five years old, microcomputing is less than twenty years old, and we're shedding the pertinent history by the dumpsterful. The history of digital computing — one of the newest core sciences in the world — is being destroyed as fast as it's being made.

Nor can we depend on the voices of the pioneers to fill gaps. Many of the originators of electronic computing, like Alan Turing, Atanasoff and Berry, John Mauchly, Wallace Eckert, Admiral Hopper, and even Bob Noyce and William Shockley, can no longer be interviewed. The British journalist Chris Evans, who wanted to be *the* popular historian of the microcomputer, died with one of his books half-finished. Computer science has become a freestanding discipline comparable in stature to almost any other physical science, and yet its public record lags far behind the evolving fact. Worthy exceptions like the BBC Press book *The Dream Machine* (reviewed next issue) only underscore the scale of the general flow into oblivion. A handful of concerned organizations, like the Association for Computing Machinery, and individual historians — Ted Nelson, Jon Palfreman, James Cortada, for example — are trying to preserve an irreplaceable historical record, and frankly fighting a losing battle.

On April 19, 1993, a few people decided to take a stand by founding the Computer History Association of California, an organization that exists to do these things:

* Create awareness of the history of computing as a real, evolving and valuable phenomenon.
* Prevent the destruction of historically significant hardware, software and documentation.

* Strengthen the cooperation among existing institutions that safeguard the history of computing.
* Begin discussions among developers and computer-related manufacturers about setting up an overall archive — or at least agreeing on archiving conventions.
* Ultimately, to help build a coalition that can build and endow a library and museum for the history of computing in California.

A tall order! But the people who attended the first meeting left, thought it over, and told friends. The idea went out in CompuServe mail and Internet mail and voice. And within days we had —

Projects.

Urgent messages about hardware slated to be scrapped, software in filing cabinets in storerooms, manuals on pallets waiting to be recycled.

Right now, today, we have almost no space, almost no money, a few members, a lot of work to do and a lot of enthusiasm. The Computer History Association of California could take off and do its part for the history of science. Or it could end up as a good idea in a filing cabinet in a storeroom. The difference is up to me, to us, and to you.

If we can make a good case for ourselves, we won't be alone. Companies and managers who share our wish to preserve this history — their history — will lend a hand to a serious effort. The wider public will contribute through annual dues or subscriptions to this newsletter.

Our potential membership is large, and growing. Computing — personally and institutionally — is moving from a devil-may-care adolescence to a maturity that embraces social responsibility. Recycling, waste control, power consumption and other "green" issues are developing broad constituencies. These people are the same ones who will recognize, in and through our Association, steps that can be taken now to prevent a lot of regret in the next century. The sooner we can alert them to the need for preservation, the more we can accomplish.

To those of you who want to share in that accomplishment, the Computer History Association of California makes five promises:

* We will be non-partisan and nonjudgmental. We will strive to be accurate, interesting and innovative. Our aim must be to enrich the history of science without distorting it.
* We will work to preserve hardware, software and documentation, as it becomes available to us, from the full spectrum of the history of computing.
* We will make the Association's property accessible to all interested parties as a professional and educational resource.
* We will aggressively pursue funding from the corporations that made the computer into a fact of modern life — inviting them to safeguard the history that they themselves created.
* We will have professional counsel on how to build up and broaden this organization; how to make time and money most effective; how to choose and manage exhibits and resources, and protect them for future generations.

If we succeed in these ambitions, we will do our small part of a big — of a great — job. We will help to affirm the history of a core science while we live surrounded by its turbulent origins. Someday, when our descendants respect the pioneers of computing as they do Galileo, Edison and Goddard, that affirmation will pay off.

But we are those pioneers. We know this story as no one else ever will. And we must keep it as our children's heritage — and as our own.

PROGRAMMING THE 1401:
An Interview with Leo Damarodas

by Roger Louis Sinasohn

[Author's note: Leo Damarodas has been a programmer since the days of room-sized computers filled with vacuum tubes. We first met in the mid '80s when we both worked for Noesis Computing Company, then known as one of the premier software houses in the Hewlett-Packard marketplace.

Today, Leo is an independent consultant, and lives on a sailboat south of San Francisco. I was recently able to pin him down and convince him to reminisce about his work with one of the earliest commercially available mid-size computers, the IBM 1401.]

When and how did you get into computers?

Let's see.... My actual first job was in 1965. I started off from high school as an electronic technician in the early '60s,when they were a dime a dozen, and in a period of three years, I was laid off 18 months. I got a job in one of the local mills, and went to school nights studying computer programming, circuitry and design, figuring that if I got a job in either maintenance or programming, that's where I'd work. And the mill that I was working at knew of my interest in computers and moved me into their office as a programmer, once I graduated.

Over the years, what have been the biggest changes in the computer industry?

(Laughs) Well, let's see. Going from vacuum tubes to solid-state magnetic core memory — this is where the expression *core memory* comes from. Another big one's interactive programming; getting away from punched cards. It affected the way I was working, anyway.

What has stayed the same?

The need for programmers. That's never changed. And I don't think it's ever going to.

Why do you say that?

Because I've been hearing as long as I've been in this business that computers would start programming themselves in the near future. It hasn't happened yet. I don't really think it will.

You don't think that artificial intelligence will become intelligent enough?

Not with the way computers are being built currently. I mean, programmers will be put out of business when computers become sentient. They're going to have to know what they're doing, and machines don't. It's as if we worried about cars driving themselves around the streets. Even artificial intelligence requires programming.

When I've been working, I've cursed my computer for not doing what I wanted it to...

(Laughs) You want my poem? Is that what you're asking for? *[Quotes:]*

"I really hate this gosh-darned thing, I think I'm gonna sell it.
It never does just what I want, but only what I tell it."

In that connection you've said that you shouldn't want it to start guessing what you want. You want it to do what you tell it.

Right.

Why do you say that?

If it starts trying to second-guess me, and it guesses wrong....if anything goes wrong with the computer, I'd rather blame myself than the computer, because to me the computer is a tool — nothing more than a glorified screwdriver. And when the tools start running themselves, then it's time to worry. Because if computers start doing what they think we want done, the next step is them doing what they think is best for us. It is getting to the point where computers become intelligent, but with present-day circuitry, no matter how many processors you hook up, it still can't think for itself. It still needs a program to run.

You worked on the IBM 1401.

Yeah.

What was that like?

Well, at the time, it seemed great, because it was the only thing I knew — the first computer I ever worked on. It was a hundred per cent vacuum tubes....the logic circuitry, the memory, everything was vacuum tubes. The addressing structure of the machine only allowed for 16K of memory, and there was no operating system in the modern sense.

The way into the machine was through reading the machine instructions. Each instruction portion was a letter, a readable character, and they made sense. Take some examples, M was Move; W was Write a print line; P was Punch a card; R was Read a card. When you executed these instructions, you didn't need an address, because the card read/punch always worked memory locations 1 through 80. And the printer always used memory locations 101 through 232 as the I/O buffer. It was a really simple machine to work with and a lot of fun in a way.

The only way you could get the machine to do something was put a deck of cards in the card reader and hit the start button. And that would read the card deck, load the program into memory, and start executing it. It was slow; there was no multi-processing, no nothing. Just a really simple machine.

So it was basically one thing at a time, and mostly written in machine language.

Basically, yeah, but we had ways around that. There was a COBOL compiler on the program, but we tried to avoid using it, because to compile a 16K program and get a program deck out would take something like an hour. But because we could read the machine instructions, if we had an error, we didn't really have to go through the compilation. You could take the program deck and modify the actual machine code, load the program and run it again. It saved a lot of time.

We also had a language called Autocoder which was a kind of assembler. It expanded the machine instructions out to more readable mnemonics, and allowed you to use labels for addresses and stuff — use real names. It made the programming a lot easier. The thing that was really interesting was the COBOL compiler, which was the only other language, that I knew about — I think there was a FORTRAN available too, but I coded business applications, so FORTRAN wasn't used that much. One of the features of the COBOL was an ENTER instruction, so that while you were writing your COBOL code, you could say ENTER AUTOCODER and start writing Autocoder code right in your COBOL source, then say ENTER COBOL and start writing COBOL code again. This was possible because the COBOL compiler didn't generate machine language — it generated Autocoder code, and then called the Autocoder, which converted the COBOL output into machine language. So it just substituted the Autocoder code in your COBOL source for the output.

What sort of applications were you working on?

Payroll, Accounts Payable, Accounts Receivable, General Ledger — straight business applications. And it was next to impossible to write any major program without going over 16K. So you either broke it down into steps — 16K steps, or you wrote program overlays. There were instructions that would allow you to read in the next part of the program. But the overlays had to be set up in such a way that you performed one step for all the data, loaded the next step and performed it for all the data. You couldn't go back and forth between overlays because the programs had to run from card decks. Data resided on disk, but not programs. All the data would come in initially on cards, be transferred to the disk drives on the system, and the programs could process disk.

So, for example, a run to print the payroll, how long would it take that program to produce checks?

Honestly I don't remember, but a long time, because the only time we could use for testing was between midnight and eight in the morning. The machine was being used the rest of the day to do production work, and about all they were doing was Accounts Receivable, Accounts Payable, General Ledger, payroll and some inventory. Not too much more than that.

What happened if you had a deck that's data to be input, and you have a deck that's a program — what happens if you mix them up? That is, you put the data in as if it were a program?

One of the things that would happen, if you tried to load the data in without having the program loaded, was that it would choke on the first card. You've got to remember there was no operating system, and this machine's just sitting there, waiting for a bootstrap program that had to be in the first card. When you pressed the start button on the console, it read the bootstrap card and branched to location one. If there wasn't a valid instruction there for it to execute, it wouldn't do a thing. The bootstrap program read the rest of the cards in, and when it got an end-of-cardfile, it branched to the location where the program was loaded; I forget exactly what part of memory that was.

Punched cards had 80 columns, so you could, in theory, have up to eighty instructions per card. Is that correct?

In theory you could, if they were instructions that didn't require addresses. They were single-byte instructions.

So a small program might fit on a single card?

Yes, in fact the bootstrap program didn't even take a whole card. The bootstrap program was about a dozen characters long. Somewhere I have a framed white poster, about four inches high and ten or twelve inches long, with that program written on it. I got it at an HP user group meeting where I walked by a booth with a sign outside that said "If you know what this is, you're showing your age." It was the 1401 bootstrap program. It looked familiar but I couldn't quite remember what it was, and I said "I don't know what it is, but I should." The guy who was running the booth knew me and my background, and he said "I know you should." Maybe three or four weeks after the meeting, the poster showed up in my mailbox.

How long did you work with the 1401?

Only about a year, and I think the machine I worked on was actually a 1410. 1410's were the ones that had disk drives. I came in just as they were doing a conversion from the 1401 series to an IBM 360.

And 360's are still being used today.

I would imagine so. *[Editor's note: Our best information is that at least two System/360s are currently used in California, both by private corporations in Greater Los Angeles.]* As for the 1401 series, last I heard, the Department of the Navy was still running an application on a 1410 a good eight or nine years ago... In the early eighties, anyway.

[Concluded next issue]

I Played the ORIGINAL Video Game!

a recollection by Scott Robinson

I went to work at Bolt Beranek and Newman (no comma, please) in the summer of 1966, as an instrumentation engineer. In those days the company's activities were roughly equally divided between acoustics — both architectural and underwater — and computer science. The computer group's main machine was a PDP-1, which consisted of about six 6 foot racks full of hardware. It may have had a Remington Rand Fastrand drum memory; I'm not sure. The company certainly had one of these beasts later on, a drum about five feet long and two

later on, a drum about five feet long and two feet in diameter with a large number of heads. All this was housed on the lower floor of the (then) new split-level building, adjacent to the kitchen and the reception area.

The control console for this machine was on the end of the row of racks; it had a monochrome CRT, about 12 or 14 inch size, and a row of miniature metal-handled toggle switches to enter data and addresses when necessary. These switches were used as the controls for Spacewar. This game was not a time-shared activity; I suspect that we used the whole machine!

When a game was started, the screen would light up with two different ship icons against a random background of stars. There could, optionally, be a sun in the middle exerting gravitational influence on matter. The gravitational constant was also players' choice, I think in two steps, "fast" or "slow." The screen was topologically connected side-to-side and top-to-bottom; if you exited screen left, you reappeared screen right, and so on.

Each ship could be rotated clockwise or counterclockwise, fire reaction engines that eventually ran out of fuel, and fire missiles of finite range and finite number. The ship obeyed Newton's laws, accelerating and decelerating under the influence of its engines and of solar gravitation, if any. Rotation could be either easy to control (when you had a switch on, the ship rotated,) or more difficult and realistic (the switches applied angular acceleration, so that rotation increased or decreased gradually depending on switch settings). The object, of course, was to blow the other ship up.

If you were hit, you were dead meat! Falling onto the sun was comparably ill-advised. Collision of two ships produced a vivid, graphically depicted explosion on screen, and both players were out, whereupon the game restarted.

For those in desperate circumstance, faced perhaps with a barrage of missiles incoming and too close to dodge, there was an escape...hyperspace! By rotating in both directions simultaneously, the ship could be made to vanish and reappear with unpredictable position and velocity. Your situation might be improved...but with a catch. The ship might explode upon re-entry into normal space, and the likelihood increased each time hyperspace was invoked. I don't think I ever saw anyone use hyperspace four times in one game without blowing up.

The display was a vector-type CRT and the quality of the graphics exceptional. The motion was perfectly smooth, with no aliasing artifacts noticeable.

Although I and others spent many enjoyable evenings playing Space War, the test word toggle switches used as controls enjoyed the game much less than we did, and failed with some regularity. Ultimately the computer folks got tired of replacing the switches and threw us off the machine. Nonetheless I take a certain satisfaction in having played one of the first computer games, an innovative and engaging game with rigorous simulation of physics in action. As for the hyperspace feature, haven't you ever felt that you were about to go off into hyperspace when you tried to rotate both directions at once?

NEXT ISSUE....

INITIATIVE 1999: Why a lot of hardware will be scrapped at the turn of the century. Why six years is barely long enough to prepare for the consequences. Plus: Programming the 1401, part 2. Smalltalk Then and Now. Palfreman and Swade's Dream Machine. More....

Downloadable October first — don't miss it!

GUIDELINES FOR SUBMISSION

The ANALYTICAL ENGINE solicits manuscripts of 600 to 1000 words on the general topic of the history of computing. Articles should be tightly focused on one interesting or illuminating episode and should be written for a technically literate general audience. Submissions are welcome from both members and non-members of the CHAC and a one-year membership, or extension, will be given for each article published. Article deadlines are the first of each month prior to publication: June 1 for the July issue, September 1 for the October issue, December 1 for the January issue, and March 1 for the April issue.

Decision of the editors is final but copyright of all published material will remain with the author.

The preferred document file format is Microsoft Word for DOS or Windows, but almost any DOS or Macintosh word processor file will be acceptable. Alternatively, please provide an ASCII file. Submit manuscripts on DOS 5.25" or DOS or Mac 3.5" diskettes, or by modem as a file attached to an Internet message. Please avoid submitting on paper unless absolutely necessary.

Can you spare a few minutes a month? The ANALYTICAL ENGINE urgently needs volunteer help for administration, subscription fulfillment, proofing and keying. Help keep the ENGINE spinning — reply to our Internet or mailing address today.

The ANALYTICAL ENGINE, newsletter of the Computer History Association of California, is published four times a year — in January, April, July and October — at El Cerrito, California. Subscriptions are available with Association membership at $25 per year. Use the coupon below to subscribe, or contact the Association at:

US Mail: 1001 Elm Court, El Cerrito, CA 94530-2602
Internet: cpu@chac.win.net
CompuServe: 72341,2763
FAX: 510/528-5138

CONTENTS

SUBSCRIBE!

and share fascinating insights into the vital story of computing while you build an organization that safeguards our unique scientific and technical heritage. Just $25 will bring you four issues of the ANALYTICAL ENGINE — filled with non-partisan, technically and historically accurate articles — and the satisfaction of preserving the history that you work to create.

____ Yes! Please enroll me in the Computer History Association of California for the next year and send four issues of the ANALYTICAL ENGINE to

____ Internet address: _____

____ CompuServe ID#: _____

____ Other gateway: _____

____ I would prefer to receive paper copies of the ANALYTICAL ENGINE. (A paper edition will be produced if the membership shows sufficient interest.) My mailing address is:

Name: _____

Company: _____

Suite, Box, Mail stop: _____

Street: _____

City: _____ Zip/postcode: _____ Phone: _____

Country: _____

____ I will submit an article to the ANALYTICAL ENGINE. (Please refer to the guidelines for submission, above.)

____ I will help produce the ANALYTICAL ENGINE or do other work for the Association. Please contact me.

** Please photocopy and mail to: 1001 Elm Court, El Cerrito CA 94530-2602 USA **