Innovations in Computational Sciences

My Work At Lawrence Livermore National Laboratory, 1976-2012

by Paul F. Dubois

March, 2022, El Cajon, California

ACKNOWLEDGMENTS

My special thanks to Bill Lokke and Steve Warshaw for reading early drafts of this work, and to my wife Barbara for editing it.

Recalling things at an advanced age is an imperfect thing, and I apologize for any inaccuracies. If I did not mention everyone I should have, it is not intentional. As I used to say to my colleagues about programming errors, do not mistake incompetence for evil intentions.

Table of Contents

Visiting my father at the Oakland Tribune, 1954

"Fasten your seatbelts, it's going to be a bumpy night." — Bette Davis, *All About Eve*, 1950

Introduction To My Technobiography

MY LIFE WAS DRIVEN BY CHANGE

My life has been lived in an era of technical revolution that spoke to me. Since technology and my career are inseparable, this work is better called a technobiography of us both. I especially loved science and electronics from an early age even though my family was not an academic one, with only one book in our house. When I was nine in 1954, the Regency Model TR-1 AM transistor radio became available at Christmas. We got a black-and-white TV, and when it broke down I loved to help my father in diagnosing the problem and going to the supermarket to use the "vacuum-tube tester," and buy replacements.

Almost as soon as I could read, I spent afternoons after school at the Montclair Branch of the Oakland Library, located just a few steps from my elementary school. I read everything in the children's section including its many books on science, such as "The Earth for Sam" about geology. The continents don't move, it informed me. Other "facts" I learned were that Mars might have water, Venus was like Earth, and Saturn was the only planet with rings[1].

When I was 12, Sputnik orbited the earth. I became a space nut, watching every televised rocket launch even if I had to get up in the middle of the night. Broadcasters like Jules Bergman and Walter Cronkite breathlessly guided us through everything from Vanguard to the Shuttle. I never missed one launch and was soon afraid to miss one, in case I was their lucky charm.

Auspiciously, the computer language *Fortran* ("Formula Translator") was discussed in my high-school mathematics class. Out of curiosity, I looked up all about Fortran II and taught it to myself. But I never had access to a computer and never ran a program. At that time, there were some large, expensive "mainframe" computers, which processed one job at a time from instructions punched into cardboard "IBM cards."

A smaller mainframe, the *PDP-10*, became available for $6500 in 1966, the year I graduated in Mathematics from U. C. Berkeley. Berkeley did have one computer in the basement of the Math Department building, but I never saw it, nor were there any courses in Computer

[1] Helping my son with his fifth-grade science, I had to look something up. Amazed he said, "But I thought you knew everything." I said that a lot of the books I read when I was his age had turned out to be wrong. He thereafter would say, "Everything you know is wrong."

Science in the Math curriculum. Computers were considered the province of Electrical Engineering. We did registration and class selection by IBM card. Every card was stamped, "DO NOT FOLD, SPINDLE, OR MUTILATE," and we students laughed that we were being controlled by machinery.

Even as I received my Ph. D. in Mathematics from U. C. Davis in 1970, I had not used a computer. No one had a cell phone or home computer, and any individual computers were not connected. But these things were coming. What we were about to experience was a rate of change similar to that of the Victorian era, with its movement from rural life to urban, and from hand-made products to assembly-line goods. It would be just a few years before the computer caught my attention for good. Between being on the leading edge of the demographic explosion called the "Baby Boom" and the changes brought on by technology in every aspect of life, it was going to be a bumpy ride.

Science Adds A New Method

Sciences had two branches— the experimental and the theoretical. In Physics, for example, there were experimentalists such as Marie Curie and theorists such as Isaac Newton. The theorists model the world with mathematics and test these models against the data from the experimental results. Unexplained data inspire the development of new theories, which inspire the experimentalists to collect more data.

From the needs of World War II to the 1970s, a third way of working in science, C*omputational Science,* was being born. Computational Science should be distinguished from the more familiar term, Computer Science, the study of computers themselves and the best ways to use them. Computational Science refers to the use of computers to solve problems in science in tandem with theory and experiment.

Here is a real example. The equations of hydrodynamics describe in theory how water will flow, but elegant solutions to these equations only exist for certain idealized situations. When features of the real world are introduced, such as obstacles in the channels or channel boundaries, it becomes much more difficult to "solve" the equations. Instead, the scientist must begin with the initial state of the fluid, say water, and continually calculate the state of the water as time advances. This requires choosing *algorithms,* specific programming steps, to approximate the solutions of the partial differential equations involved.

By such means, a computational scientist can understand whether or not current theory is adequate, solve real-life problems, understand the results of experiments, test designs for

new experimental machines, and simulate experiments that are infeasible because they are too dangerous, too expensive, or require inordinate complexity.

Algorithms can include making decisions about the calculation based on the values of the data. In addition, each program needs a supporting environment on the computer in which to write the programs so that they are easy to modify while maintaining correctness, and which provide a powerful interface for the user/scientist to work, individually and in groups.

Lawrence Livermore National Laboratory

After six years teaching at University, I was excited to be hired at Lawrence Livermore National Laboratory in Livermore, California. The Lab had been founded in 1952 as a facility for research and development work on nuclear weapons and related sciences. An excellent book by C. Bruce Tarter, who was the Lab's Director when I retired, details the history of the Livermore Lab through 2007.[2] A separate history of the computers and the nascent field of computational physics[3] was written by William A. ("Bill") Lokke, who played a key role in the Laboratory including as Tarter's Assistant Director. These two physicists were my mentors and supervisors at some level for most of my 30 years there.

Although some computing had been brought to bear on science, its real birth took place when rapid hardware improvements began. As the power of computers doubled every 18 months, so did the ability to accurately model more and more of the world and to complete labor-intensive tasks. Such successes encouraged more investment in hardware and software technology, and the boom was on.

Picture The Way We Were

In the world of 1976, the IBM Personal Computer is still five years in the future. Tim Berners-Lee will not come up with the concept for the Web for thirteen years, and it will be eighteen years until it is in general use. Tape drives to store data are six feet high. Telephone modems transfer data over phone lines at a few characters per second.

Remarkably, the Livermore Lab was pioneering *time-sharing*— a computer dividing its attention among multiple jobs. Before time-sharing, a computer processed one job from

---

[2] C. Bruce Tarter, *The American Lab: An Insider's History of the Lawrence Livermore National Laboratory,"* Baltimore, Johns Hopkins University Press (Johns Hopkins Nuclear History and Contemporary Affairs), 2018.

[3] W. A. Lokke, *Early Computing and Its Impact on Lawrence Livermore National Laboratory*, UCRL-TR-226840, Lawrence Livermore National Laboratory, Livermore, Calif., 2007.

beginning to end, or perhaps to a stopping point. With time-sharing, users could see their computations advance a bit at a time, without noticeable interruptions. This meant in particular that shorter jobs could complete without waiting for a very long job that had started earlier, and development could take place interactively. Large jobs were run at night using a batch system.

For the programming environment, we were advancing from the keypunch and "decks" of IBM cards to re-purposed Teletype machines, and time-sharing on the mainframes or on smaller machines such as the PDP family.

Each scientist at Livermore had a Teletype keyboard but to avoid having to repeatedly print out programs that had changed, and to be able to view graphs, Livermore had another experimental facility— the "Television Monitor Display System" (TMDS) which displayed the text of the program being developed, and program output and graphics. The monitor resembled a large television set hung down from the ceiling. Because of their limited numbers, each two adjoining offices shared a TMDS channel: what appeared on one scientist's set also appeared on another's.

At that time, scientists programmed almost exclusively using the Fortran language, but for some considerable time we also required use of the *assembly language* of the latest mainframe ("*supercomputer*") to achieve maximum speed and memory use. The first task when receiving a new type of machine was to rewrite those components in the new assembler language, which was documented by the manufacturer. In the beginning, we had only a text editor which required typing in commands on the teletype. Visual editing on a screen with a mouse was years away.

By the time I retired, we used local "workstation" farms and had phones in our pockets which had much greater capacity than the supercomputers of 1980, used electronic screens, and logged into supercomputers having many computational *processors*, soon to number in the hundreds of thousands and even millions. We talked about *petaflop* computing— that is, 10 to the 15th power FLoating-Point Operations ("*flops*") per second. Today they talk teraflops, 10 to the 18^{th} power. Much of management focused on having the largest and the fastest hardware, and if Livermore had a weakness for hardware but devoted inadequate resources to software that got the most from the computers, it was not alone, as every institution wanted to be the fastest in theory if not in capability in real applications.

Supporting Computational Sciences

During the period that I was a Mathematician/Computer Scientist at Livermore Lab, my principal contributions were:

1. Developing the concept of Computational Steering— blending a powerful, interpreted user-interface language with fast compiled-language calculations.
2. Inventing the Basis System, the first widely successful computational-steering program-development system.
3. Helping design Numerical Python for the Python language, and acting as its open-source steward;
4. Introducing and promoting the use of Object Technology for scientific programming;
5. Writing and teaching about scientific programming and software engineering, including editing the "Scientific Computing Department" of *Computers in Science & Engineering* (formerly *Computers in Physics*).
6. Developing and implementing techniques to assure program correctness in an atmosphere of continuous change.
7. Improvements in the capability, speed, and correctness of many physics simulations.

These pages describe how I came to be at Livermore Lab, my experiences that led to the invention of Computational Steering (Basis), and how our team eventually revolutionized Livermore's premier two-dimensional inertial confinement physics program using Basis and enabling it to run on scientific workstations as well as the overloaded supercomputers.

I also played an important part in developing and supporting the *Python* language as a scientific tool. For three years I worked improving software for understanding climate models. My final team development, *Kull*, put a lot of my ideas to work. Kull is a three-dimensional, massively-parallel, Python-steered physics application with an automated testing engine.

I give my thoughts about what makes or breaks software-engineering tools and project management, and also some vignettes about the people that participated in this work. The Appendix describes my professional life and lists my publications.

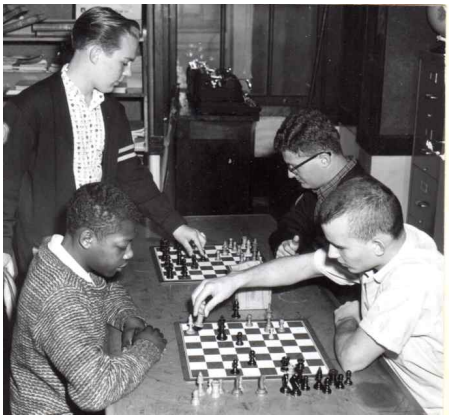
I. STUDENT AND PROFESSOR

"Twenty years of schooling and they put you on the day shift." — Bob Dylan

GROWING A MATHEMATICIAN

I Grow From K-12





My childhood from kindergarten to the start of high school corresponded with the 1950s. A lot of it was spent on my own and in the library. Numbers and games fascinated me from the start.

One summer around the third grade I wrote down a list of all the numbers from 1 to 10,000 to amuse myself, and I used dice to generate wins and losses for baseball teams in a fantasy league I set up for the Pacific Coast Baseball League. I was excited when I figured out how the newspaper was calculating "Games Behind".

I loved Chess and Monopoly and card games like Canasta, but since my brothers were too young to play with me, and only a few children lived near me, I played against myself. I seemed to be good at arithmetic, to the annoyance of other students, but I didn't see any real

significance to this ability. I was that kid who prompted teachers to say, "Now let's not always see the same hands."[4]

I went to the library to read most afternoons, and avidly followed the space program from my teen years until present. I played "simulation" games, reading the long rule books and playing both the winning and losing side. I played "Gettysburg," "Tactics", and "Stalingrad" over and over. I put together models of battleships, especially the U. S. S. Missouri, since my grandmother could not remember which ones I already had but knew the Missouri had a lot of pieces and kept me quiet.

The space race fueled my education. The United States began an effort to encourage people of talent by testing all of us. I was placed in special classes in English, Chemistry, Physics, Electricity/Radio, and a shortened Mathematics sequence that taught four years of content in three years. Therefore, in my senior year, I started Calculus at U. C. Berkeley, to which I drove before going back to Skyline High School in Oakland. There were no Advanced-Placement classes in high school then; I was just admitted as a high-school student.

I Survive U. C. Berkeley

My family is half Belgian-immigrant and half Oregon-Trail, and I was one of the few in my large extended family to attend college. I financed my education with jobs such as cooking in the middle of the night in the lunchroom of a Dickensian glass-bottle factory. I also worked in the University cafeteria, at a city summer camp, and in my Fraternity. I survived the captivating but disruptive "Free Speech Movement" in Fall, 1964, when a number of students skipped classes. There were also many demonstrations against the War in Vietnam.

As in any University, many academic and cultural speakers appeared, but I was too busy working to attend. My future wife and I did hear Sargent Shriver, who invited students to join the new Peace Corps.

I hated Berkeley. The faculty were often abusive, and made it clear they didn't want undergraduates to bother them. To cite just a few examples,

- In his last year of teaching, the Mathematics Department Chair taught us by reading his own (expensive!) book word-by-word, copying diagrams to the blackboard with any mistakes they contained intact.
- One algebraist was assigned the Real Analysis class, prerequisite for the rest of the Upper Division, but announced he was just going to teach the content of the other

[4] Once the fifth grade teacher banished me from math class to sit outside on the porch in a heavy rain, giving me pneumonia.

prerequisite, Abstract Algebra, and we could go learn Real Analysis on our own. They promoted him to Department Chair.

- On entering as a high school student, the head of the Physics Department signed me up for calculus for engineers, not for mathematics majors.
- In my third year my assigned advisor told me I should quit the University because I'd gotten a C in one course.

I later asked a member of the next generation if the faculty had improved, and she said it was still the same.

Though I completed the program for majors in Mathematics, it is relevant to what follows to say that I also completed a minor in Social Sciences, with courses in History and Sociology. The Social Sciences curriculum itself was not science as I knew it. I got good grades by putting matrices in a paper or two, leading to the professors being eager to recruit me as a grad student to put matrices in their papers.[5] The only problem was that in Sociology, it would take 12 years to get the doctorate, they said, because they required each student to undertake social work in the middle of the program, so that you would be down with the class struggle.

Graduate School

I finished Berkeley in January, 1966. I could not just get a job then because I thought I would be drafted. I married a high-school friend, Barbara Russell, and we were both accepted into Peace Corps training and went to India. We came home from an unsuitable program there so that I could start graduate school at U. C. Davis.

The grad-school curriculum in Mathematics was purely theoretical and without computation. It was tough. I was the only student out of ten in my class who passed all four subject qualifying-exams and finished the Ph.D. in four years. One other student finished in five. My thesis was in commutative ("Abelian") groups and my thesis director was the kind and inspiring Dr. Doyle Cutler, a laconic Texan out of New Mexico State University.

We had a good time at Davis, the opposite of the horrible experiences we both had at Berkeley. In Central California we experienced seasons for the first time, the amazing veterinary school saved our cat, and I got to teach my first class as the professor on my own.

[5] In 1980 I did use my training. I co-authored a paper with my brother Phil that appeared in the political science journal, *Polity*.

I discovered that I loved teaching. But I still could not get a job without being drafted. So my wife and I sought a life elsewhere. Just before leaving, I flunked the draft physical.

UNIVERSITY OF ALBERTA: I GET COMPUTER TIME

I won a two-year I. W. Killam Postdoctoral Fellowship from the University of Alberta, Edmonton. The Killam is an interdisciplinary Canada-wide program. My supervisor was an algebraist, Dr. S. K. Seghal. I started there by writing up my thesis for the *Transactions of the American Mathematical Society*, and I published one paper with Seghal. After only a few months, I knew that I enjoyed teaching and I loved Canada, but I didn't love doing abstract research in Mathematics. I'd been fooled by being good at it and by the pleasures of working with Dr. Cutler at Davis.

Instead of writing papers in pure mathematics, I taught and worked on two projects in mathematical biology. For a study of dandelion populations, I needed to write a program. My office mate, Ray Holmes, an unusual hippie-appearing mathematician from Dalhousie University, was already into working with APL ("A Programming Language"), a bizarre language full of symbols. From his desk behind mine, Ray would hand a piece of paper over my shoulder with one long line of APL on it. "Guess what that does?" he would say. Inevitably the answer was something amazingly complicated like "solve the diffusion equation".

With Ray as a guide, I decided to write the dandelion simulation in APL. It worked, but it ran quite slowly, since APL is "interpreted", and the computers were slow in those days. I went to our Department computing center, and the manager there, Mary Willard, got me started in Fortran for my project. The simulation ran, and we confirmed a biologist's theory about dandelion populations. I also studied the distribution of parasites in duck guts; ducks are big business in Canada.

Mathematics, as with the other sciences, has a theory and an experimental side, which they refer to as "pure mathematics" and "applied mathematics". I no longer had an interest in pure mathematics. It certainly wasn't as interesting as working on real problems or developing simulations on the computer. But that made me a pariah in academic mathematics, which is nearly universally "pure". The word they use, "pure", tells you anything else is "impure".

My wife helped me write and send out 300 inquiries to look for academic jobs. I got one interview at New Mexico Highlands University (NMHU), but I did not receive an offer. The

OPEC Oil Shock had wiped out new jobs and the supply of gasoline. I turned down one other idea— teaching in the prison in Vancouver.

The University of Alberta had me stay one more year as a Visiting Assistant Professor. I accepted student work in either French or English, but I didn't notice decent writing in either case. Still I wasn't getting any job offers. I'd started looking for non-academic jobs. However, the chair of NMHU, Prof. Kim Kirkpatrick, called to offer me the position that I didn't get the year before. So we gave away our ice skates and toboggans, and we were off for Las Vegas, New Mexico, the "other" Las Vegas.

COMPUTING IN NEW MEXICO

I should have asked why the previous professor had left NMHU after only one year during a job market devoid of openings. I taught at NMHU for three years but was more or less miserable professionally. The educational preparation of students there was not what it should have been. I was trying to help students get to a college level when they didn't know anything about signed integers. This was a "bridge too far".

During my time at NMHU the Mathematics staff "upgraded" the University's IBM main-frame to 48K of main memory. The "K" is not a misprint. The computer occupied a cabinet about six feet by four feet by three feet. It had to be cabled under a raised floor to provide cooling, but during winter it got too cold, and sometimes the printer wouldn't work. Students submitted their programs as stacks of punched IBM cards, and student employees would run the jobs through, one at a time. If a program had an error, it might be hours before the student could repair it and run it again. The University had hoped to replace their retiring bookkeeper, Mrs. Ula Comerford, who had all the tax tables, etc., memorized. In the end the school not only paid for the new computer but had to add two new bookkeepers to replace her.

I taught courses on Fortran and IBM's short-lived PL/1 language, and a special course on simulation, my long-time hobby, but the projects for this course were not on the computer since knowledge of computer languages was not required. Since I now understood how computers work, I took my turn as supervisor on the evening shifts running students' punch cards through the computer. When the computer failed, there was a "reset" button. I was horrified by one of the student workers who kept punching the reset button before reading the indications of what had gone wrong.

I also taught Statistics for non-majors. A big problem for these humanities or education students was that calculations were frequently needed, including square roots. That involved

a complicated process when done by hand, especially for students who were not great at arithmetic. I arranged for the Math Department to buy 25 of the new Texas Instruments hand-held calculators, which had four functions and a square root, and we rented them to students for $5.00. The calculators paved their way to success. A full class finished the course, including my wife, who got an A.

But this success could not fix the local schools. I put a question on a final one year that required a short written answer about the applicability of a statistics test. The written answers were virtually incomprehensible in either English or Spanish. I bought a Sympathy card and sent it to the chair of the English Department.

Technology called again, and I bought myself a hand-held calculator, the first *programmable* Hewlett-Packard scientific computer, the HP-55. It used "reverse Polish" notation, and I could key in arithmetic or function processes of up to 49 steps in length. I paid $400 for it, about 4% of my yearly income. Excitement such as this continues to keep me an "early-adopter" of personal technology.

My wife worked in the University Library and took classes to finish her B.A. in English with a minor in Music.

Ralph Carlyle Smith

At NMHU we met an interesting person, Ralph Carlyle Smith, who was the University lawyer. He told of being called to the middle of the Pentagon while it was under construction during the war, then being sent on a train to Santa Fe. He had been told to retrieve instructions at a post-office, not knowing what he was going to be doing but that it was for the war effort. He learned he was going to Los Alamos. He said he was not allowed to tell even his wife where he was, and being abandoned, she soon divorced him. (At least, that's how he told it.)

Ralph's work was as a patent lawyer for the Manhatten Project., a concept which at first thought seems odd. He went to meetings where the physics people talked about what they were working on, and he took notes. However, he said, Edward Teller, the future founder of Lawrence Livermore, would get into excited discussions with Hungarian physicists, and they would switch to Hungarian.

LOS ALAMOS: ASSESSING ENERGY SUPPLIES

In Alberta I had joined a group that played the Asian game of "Go", and when I got to Las Vegas, New Mexico I sought out the Go players in Santa Fe. Most of them were employed

at Los Alamos National Laboratory (LANL). These Go friends heard my complaints and steered me toward a summer position at LANL as visiting faculty.

The atmosphere was very comfortable for me at LANL. I worked in Burt Wendroff's group on a project called the Regional Energy Assessment Project, a response to the formation of the OPEC oil cartel. I learned how to program and use Control Data Corporation's "6600" and "7600" supercomputers, and I worked hard on learning more computer science at home.

One memorable day, someone led me to an old computer in the Los Alamos basement that was based on vacuum tubes rather than silicon transistors. They were keeping it as a potential museum piece, I think. To program this beast, one had to create a "paper tape" to be read into the computer. To update the tape, it needed to be spliced and stuck together, similar to splicing movie film. I carried out the ritual that programmers do when first using a new kind of computer by writing on the tape a program that printed, "Hello, World".

During the next year I could log in to the Los Alamos system from my home in Las Vegas, using a telephone modem, which was a cradle into which you placed the telephone receiver. The New Mexico phone infrastructure was poor, and five or ten characters per second was the limit. It was therefore popular to program using just two letters for the program's "variable" names.

Although the whole system was slow to use, it allowed me to be involved with a large computer and its associated culture. The Los Alamos connection also allowed me to hear of a summer institute at the Livermore Lab being held for minority-institution faculty, for which NMHU qualified. As my wife was still busy in school, I jumped at the chance.

The author starts at Livermore Lab, 1976. The machine behind me is a micro-fiche reader, not a flat-screen terminal, which didn't exist yet.

WORKING FAST IN LIVERMORE

My assignment during that Institute was to work in the Numerical Mathematics Group to analyze an algorithm that had been proposed as a way to solve a certain problem on a "vector" computer. These were computers that were much faster at manipulating long streams of numbers rather than repeating single operations. If given two lists of 10,000 numbers each, a vector computer is very much faster at adding the corresponding components using special "vector instructions" rather than looping over their contents, adding one pair at a time. Livermore was awaiting its first computer where vector operations would be available in hardware, the CDC STAR-100.

Algorithms were going to be needed that "vectorized" the computations to take advantage of the hardware. Some genius had figured out a way to exploit the design of the CDC 6600 and 7600 computers to get much improved results with "vector" computations, even though this had not been a deliberate feature of the computer's design. Since we could use the 7600 to explore what would happen on future vector computers using this *vector library*, I was able to study vector algorithms that summer. My summer supervisor, Gary Rodrigue, gave me the problem and turned me loose.

Using the Livermore's 7600s I was able to do calculations on very much larger problems than the author of the algorithm had been able to do, and I quickly noticed that the algorithm would fail to get the right answer if the problem was large enough. Given this inspiration, I established the flaw in the algorithm by means of pure mathematics. In effect, computational-mathematics-me had helped theoretical-mathematician-me.

Since I had previously used the same operating system and similar computers at Los Alamos, my first computation was running and displaying on my TMDS very quickly. I heard the mathematician next door, with whom I shared the channel, yell, "Is that you already?"

No doubt that speed was possible in part because I actually knew how to type with two hands instead of the two-finger style most men used back then.[6] Near the end of the summer, when the group leader of the Numerical Mathematics Group, Fred Fritsch, offered me a full-time position, I thought my typing speed had been a factor.

I returned to NMHU and broke the news to them as soon as the Livermore offer was official. The President of the school told me solemnly that leaving academia would "ruin my career." We were glad to go back to the Bay Area, as all our families lived there. There was just the little matter that I was now to work in a completely new field, Computational Physics.

[6] In high school, I dropped French after two years because of having a mentally deranged teacher, and I took typing instead. My mother said, "Take typing. You'll always be able to get a job." The class had two boys and forty-eight girls.

II. COMPUTATIONAL SCIENCES

"Find a Need and Fill It"— the motto on the Kaiser Cement trucks

LIVERMORE: NUMERICAL MATHEMATICS GROUP

An Outsider With a View

I already had the necessary security clearance from my work at Los Alamos, and I was able to begin work immediately.[7] My first office was in the World War II-era barracks, perhaps even the one where my father bunked as he trained as a Naval pilot 33 years earlier. He later got an ear infection in the swimming pool where they practiced survival, and he was washed out, since no class could wait for someone with an illness. He later became a cypher clerk on an LST and was at the Battle of Okinawa. He was also a reporter for the Oakland Tribune, and later Director of Public Relations for the Port of Oakland, ending as Director of Air-Traffic Development for the Oakland International Airport. When I was in high school, I helped him prepare the usual deceptive population charts for submitting to the Civil Aeronautics Board, using what I learned from the book, *How to Lie With Statistics*.[8]

The Numerical Mathematics Group (NMG) in the Computations Department was in some difficulty as I joined it. Like any group designed to serve a wide constituency, it needed to distinguish its own interests from that of the customers. Mathematician Alan Hindmarsh had an international reputation for his software that solved ordinary differential equations (ODEs). However, he had many more users outside the Laboratory than inside. Because of the nature of traditional programming languages like Fortran, it was difficult to adapt that software to work within the context of a large simulation. It was great for solving a set of ODEs, but it was prohibitively difficult to incorporate it into a large system in which it was only a piece. This was not Hindmarsh's fault. Years later, as the Object-Oriented revolution began, I began to understand why that difficulty existed.

---

[7] Many new hires spend months in an unclassified area called the "Cooler" while they await their clearance.

[8] Huff, D., *How To Lie With Statistics*, W. W. Norton, NY 1954; reissued 1993.

When budgets tighten, such centralized support groups may lose funding, as they may not be considered important enough to the funding group. In fact there was a popular book of numerical algorithms, and many physicists would just type in these algorithms rather than look in our library. Alas, some of the algorithms in the book were inferior to ours, and we had quite a few people come to the group wondering what was wrong.

And budgets were tightening. A consequence of buying leading-edge computers was that the customer bought bare metal and was required to supply everything else. In the computational area, there were groups in the Computations Department that supported them: an operating system group, a compiler group, a graphics group, NMG, and statistics, and the people who ran the computer center and networks. The Weapons budget had to support them all.

About two weeks after I began work, Fred Fritsch said he had lost the funding he had hired me with, but not to worry, it was his responsibility, not mine. The main driver at Livermore Lab was the Weapons program, and Fred said that they needed to add an underground test to the schedule for that year. While Weapons Program leaders knew about NMG, we played no active role in their simulations beyond providing basic mathematical software such as special functions.

My supervisor from that summer, Gary Rodrigue and I set out on a mission to get NMG involved in the big programs. He arranged my first assignment, to help the MEG code group in Theoretical Physics (“T-Division”) vectorize a certain calculation. As I did that assignment, I started hanging around T-Division when I could, learning about their projects. I wanted to offer myself as labor to improve the speed and accuracy of their “codes”, the name they gave, quite unique then, to the programs they wrote. Each code was produced using a custom infrastructure for building it and running it, so to work on a particular code required learning this infrastructure first. I give more details about the extent of this situation in the discussion “Environments for Large Programs ”, but for now we need only know that the learning phase was both substantial and idiosyncratic.

The Weapons Program contained several divisions, notably A-Division (fusion research), B-Division (fission research), and T-Division (theoretical physics). Each of these had several code groups as well as their subject experts and experimentalists. Over time, I also began to help code groups and scientists in several other Divisions.

MEG Required Both Technical and Social Work

My initial project with Theoretical Physics (T-Division) involved a code called MEG, and inside it there was a subroutine which repeated a certain calculation many times during the

program. Measurements had shown this routine was using up a great deal of computer time, and they asked me to see if I could "vectorize" the algorithms.

Historically the physicists coded their own work or sometimes worked with an application programmer. I was asked to do this project under the guidance of T-Division's leader, physicist George Kramer, because this particular physicist was a lone practitioner with only basic computing skills.

I did well on the software, but tripped up a bit on the social aspects. Besides using the vector library, I coded using the advanced features of Livermore's version of Fortran, called LRLTRAN (the Lab had formerly been named "Lawrence Radiation Laboratory"). For example, I used the "IF-THEN-ELSE" structure that was going to be in the next Fortran standard but which was not yet common, rather than the unreadable "arithmetic IF". When I delivered my new routine, twice as fast as the old one, both George and his technician Roy Swiger said I needed to get rid of that "new stuff". I said no, it was a major improvement, was in the coming standard, was already supported by the Livermore compiler, and they could get used to it. They relented. The arithmetic IF sooned disappeared from history.

Alas, the physicist threw up his hands and said he could never accept this; he could not work on it with all these new concepts. In the end he was ordered to use my new version. This incident showed me that implementing improved methods was going to be an ongoing "situation" for scientists who worked primarily in their subject, with computations technology being an afterthought. Fortunately, T-Division was happy with me, and I did more work for them.

I Uncover Unfortunate Coding

The theoretical scientists also did not know about "memory management" yet. The 7600 had a limited core memory (256K) and a larger "extended core" that was essentially a semiconductor disk that the programmers had to laboriously manipulate.

Each algorithm needs temporary space to hold intermediate calculations. At that time MEG authors would extend the size of MEG in main memory,[9] use some of it to allocate this temporary space, and then snap back to the size before. I discovered that the developers were reusing some of the "temporary" data later in time; they relied on the fact that they "knew" nobody had written over the memory that far out in the meantime. This was a disaster

[9] A sociopath in A-Division discovered that if he expanded his code to consume the entire memory it would exploit an error in the operating system— his program would run to completion, never letting any other program take a turn. When caught eventually, he was proud.

waiting to happen. I researched how to write a real memory manager called a “Heap” manager, and revised MEG to use it.

I also discovered that MEG had about 23 versions of Avagadro’s number coded into it in various places. Each developer, working on a section of the program, entered an approximate value where needed. The developers currently didn’t see the need to use the same value all the time themselves, much less see what others had used. Sometimes old Avagadro was disguised as half of himself, or raised to some power. I suggested a way to fix this.

A “source-code control system”, or SCCS, is a system for exactly tracking changes by a team, and merging together changes. These days, there is an entire industry producing SCCS software. Much later in my career I was able to convince people to use an SCCS. At the time I worked on MEG, its SCCS was a technician named Roy, whose title was The Centurion becaused he guarded the source.

Not to pick on the MEG group, though. As I learned to use more and more of the Lab’s large programs, they were all in more or less the same state. Math routines and one database format were all that was available for sharing between scientists or groups. There was a central Computations group that made a library of graphics routines for display of data, but the library was not particularly designed around the needs of the scientists/customers, but rather the needs of the graphics group to produce something state of the art. Making the simple curves and contour maps the scientists needed was not interesting to them. As happened in other situations, this led to some scientists writing their own graphics routines.

The STAR fails

When the STAR arrived, Gary and I visited it shortly after it was installed. It had an incredible number of blue and white-striped connecting wires along its eight-ft length. I thought, “How could all those connections have been done correctly?” I was probably remembering my laborious after-school job in a factory hand-testing cables to connect Polaris missiles to submarines.

Indeed, the damned STAR never was reliable. I had one frustrating year on the STAR trying to build a program that took 30-minutes, while the machine had a mean-time-to-failure of 25 minutes. It was quickly abandoned at the well-timed arrival of the Cray-1. The work on vector algorithms paid real dividends on the Cray.

* * *

CONTINUING PROGRAM INVOLVEMENT

Working With Steve Warshaw

From 1979 to 1982 I continued my foray into the halls of the physicists, and found a working partner in Steve Warshaw of T-Division. Together we worked on algorithms for the management of moving meshes, and on Equation-Of-State calculations (EOS), the part of a physics simulation that connects it to the actual behavior of real materials. An entire division of materials scientists is devoted to gathering the data for the properties of these materials at all extremes of temperature, pressure, and density. The data is produced by a patchwork of theory, experiments, and simulation. Hence, our project motto, "EOS, what a mess."

Steve and I worked on speeding up such calculations and figuring out how to invert some tables in vector mode. This was immediately applicable in the Program codes. I was also able to pay back a debt to the program for minority-college summer faculty by directing a visiting professor, the late Manuel Keepler, former Chair of Mathematics, South Carolina State University.

Ionosonde: Important Success for Steve and I

Then Steve was asked to take over work to support verification of international treaties, the *Ionosonde* Project. The work was to predict the movement of the ionosphere in response to sound waves such as those produced by ground movement or surface explosions. A radar would record the Doppler shift caused by electrons in the ionosphere moving up and down as sound waves passed by. Steve produced the equations, albeit with difficulty because the prior owner of the project would only give Steve the paper he had written and would not reveal how he had derived the equations. The person was a manager now, so Steve re-derived the equations.

My part was to develop an algorithm to solve those equations and write a program to do it. I was now half-time with Ionosonde, and there were another 10.5 "heads" at Los Alamos on Ionosonde. We were able to calculate what we thought would happen, and even made a movie displaying our prediction for an event. We were excited. It remained to show it to the program managers and await a chance to compare our prediction to experiment.

Not so fast. Our project manager, Bob Kuckuck,[10] was dissatisfied with our movie. Producing a movie using the computer was quite a new thing to do, and we were very proud of how it showed off our prediction of the physics. Bob saw that there was only a minimal header on

[10] Bob had a long and distinguished career at LLNL and at the Federal level once he got away from me.

the film telling what was being calculated. Bob wanted a Livermore logo and credits before you viewed our "big bang" result. I had someone in the movie department fix it. Bob was thrilled.

In Photostore We Trust

Accessing tapes containing previous work was slow, since the developer had to request an operator to get them from a vault and mount them in the cabinet. So I stored my work in the IBM *Photostore*, the ultimate Rube Goldberg contraption. It stored files by exposing photographic film, developing the film, cutting it into pieces and placing the pieces into plastic cubes. Then, a robot arm grabbed the cube and took it to a storage area and shelved it. When the record was needed later, the robot arm retrieved the cube, scanned the film to read it, and converted it into text. A person could be forgiven for not trusting it.

When I began next to work with physicist Steve Warshaw, he was paranoid about the Photostore failing. He demanded that I have the small code we were working on keypunched once a week, and that I give him the box of cards to store in his office. It fit into one card box, about 18 inches long. Gradually, a pile of these card boxes grew up next to his doorway. Next to this pile, across from his desk, was a massive safe that held classified documents.

On January 24, 1980, at 11 A.M., I was on the 5th floor of Building 113, on my way from Steve's office to the elevators, when the building began shaking. The Greenville-Fault earthquake, with its epicenter just a few miles away, was a significant long shock of magnitude 5.5. Within two minutes there were two aftershocks of magnitude 5.2 and 4.2. Some office trailers fell off their moorings. The book-stacks in the library fell like dominoes. We learned later that our building had not been that far from collapsing, as it consisted of a central core for the elevators and four outside posts, with the floors suspended between the core and the posts. The lower floors had the computers, with offices above them.

I sought the nearest doorway as a native Californian would do, but it later turned out that the doorways had no headers and were not strong. As the shaking stopped after a long 40 seconds, I rushed back to Steve's office. He was just coming out from under his desk. He had not gotten under his desk on purpose; the initial jolt had knocked him out of his chair. He started to get up and saw his giant safe advancing across the room toward him, so he stayed put.

The entire history of our new program was spilled out in multiple layers across the floor in IBM cards. Steve, somewhat in a daze, and as white as a sheet, said "I didn't like that, Paul." I replied that we needed to get him out of there and get out of the building. He didn't understand. "There will be aftershocks," I said. Realizing that almost everyone on that floor

was a physicist from the East Coast, I quickly went around rounding them up and leading them down the stairs.

Had the building collapsed, it would have crushed the computers, the tape vault, and the Photostore. Important codes got stored yearly elsewhere, but ours was not that important yet. The Lab had made the Fukushima mistake, having the backup at risk with the primary. But I suppose if Steve and I were gone with the building that wouldn't have mattered.

Operation Mill Race

To further test a variety of projects, including our work, the DOE engineered a large surface explosion at White Sands, N. M. by detonating a giant pile of ammonium nitrate and kerosene, a so-called "fertilizer" bomb, and was called "Operation Mill Race". I was asked to decide where to set up radar equipment to record data for Ionosonde, since I knew New Mexico. I chose one location near Santa Fe. Other equipment was placed elsewhere. It turns out that sound waves can propagate long distances since the density of the atmosphere varies at different heights and different locations. It is possible that a large sound can be heard at one place but not at another place closer to the source.

When the data arrived and I plotted it against our prediction, I was crestfallen. The shape and timing were right, but the amplitude was off by a factor of two. I took the graphs to Steve and Bob and expressed my disappointment. They looked at the data and said, "Wow, this is fantastic!" I was shocked to learn that a factor of 10 on the amplitude would have been good enough for them. I suppose experimental science is not as precise as you would hope.

Our Education in D. C. Classification

We sent a movie of the predictions and the field data to DOE in Washington. When it came time for a review, we needed to fly there to report. Our presentation would be classified, so Steve had to go to "classified-carry" school. Our presentation was printed on plastic overlays to be used on a projector, now a Stone-Aged communications process. Steve double-wrapped the media, the inside being an envelope stamped with the classification, the outside the proverbial plain brown wrapper.

The flight to D.C. was late. We didn't arrive until about 1 A. M., and waiting for us is a very grumpy FBI agent who was to take care of the envelope until the morning. He is so grumpy he won't show Steve his badge like Steve was taught in the classified-carry course, but Steve relents after looking at his scowl. Next morning we arrived in the lobby of the Forrestal

Building, and there was a different FBI guy with our envelope. To our surprise, there is only one guard in the lobby. We made our own way to the second floor. The meeting seems to be in a room behind a door with a frosted glass panel, and with no guard outside.

We gave our presentation in the morning, and it was a big hit. Los Alamos, with 8 times our effort, had little to show. One reason was that their culture was different than ours. At least back then, working on a project at Los Alamos could mean doing something that interests the scientist but has only a vague relationship to the goals of the project. At Livermore, you worked on the problem for which the government needed answers.

At lunchtime, we looked around for the FBI— not there. We ask what do we do with our materials. “Oh, just leave them here, I’ll lock the door,” said the Federal program leader. And he did, with a simple key like they used for interior house doors a hundred years ago. This man had once been a radar technician and seemed to us to be a political appointee. Bob told us to speak to his level, and to use radar analogies.

One secret aspect of the program apparently was leaked by someone in the DOE or Livermore management chain in an attempt to get more money for the project. This occurrence angered one external partner. Over my career, I needed to know some secrets, and except for actual blueprints and code, I believe that quite a few of those secrets were subsequently leaked by someone from the political class up to and including the Congress and the White House, either trying to get a project funded or trying to kill another project to get its money.

More than ten years after the Ionosonde project, after the fall of the Soviet Union, Steve Warshaw invited me to see a talk at Livermore Lab by Jana Drobzheva from Kazhakstan, about her research on ionospheric sounding. After her talk, Steve and I introduced ourselves. “Warshaw and Dubois!” she exclaimed. “I’ve had your paper[11] on my desk for three years!” Steve recently told me, “She is also the daughter of the Premier of Kazhakstan. I have been in touch with her and Valery Krasnov since their visit to the Lab back when. She and Krasnov are now based in St Petersburg. They authored several open-literature papers on the ionospheric sounding of acoustic pulses from the Russian underground explosions.”

This illustrates the tension in scientific work on sensitive topics. Just keeping everything secret has unfortunate consequences. The people involved become invisible and won’t be able to get a job outside of secret work later; knowing that, many of them won’t agree to be hired. If you try to pay them extra, there are complaints that you are paying too much compared to “industry surveys”. You have to put up with this problem for some of the

[11] P. F. Dubois and S. I. Warshaw, “Preliminary Theoretical Acoustic and RF Sounding Calculations for MILL RACE”, Lawrence Livermore National Laboratory, Livermore, CA, UCID-19231 (1982)

employees, such as weapons designers, but it won't work in general, as I discussed in an article about the social contract with scientists in important specialties.[12]

For example, some of our physicists left when times were tight to go work for the financial industry. As it turns out hedge funds and scientific simulations have a lot in common. It they had not been able to point to at least some public evidence of their prowess, their careers would have entirely depended on the volatile funding of the government. They are amongst the smartest people on the planet, and they aren't going to agree to a life like that.

As I moved on to work at Magnetic Fusion, I helped Steve find personnel for his growing Ionosonde team. It grew to include five or six people. Steve works on these ideas even in retirement[13].

Seven Managers Per Capita

The Livermore Laboratory has a management structure called a "matrix". Most of the scientists had "home organizations" related to their expertise and were managed by their peers, but were assigned to work on projects managed by a project team. The idea was that projects would come and go, but the expertise-level groups would be able to supply people as needed. The expertise organizations reorganized constantly.

The *Ionosonde* Project's manager, Bob Kuckuck, was part of L-Division, under a Treaty Verification program. Steve Warshaw was in T-Division, part of the Physics Directorate. My home directorate was Computations, and I had a Group Leader there and a Division Leader and so on, up to the Associate Director. Associate Directors are like Knights, and they all convene in armor around a Round Table but basically run their own fiefdoms.

The upshot to the scientist is that one year, Steve and I were exhausted by giving reviews to all these chains of command. We counted up how many people we were managed by, and it came to eleven people, who supervised 1.5 people.

Rearranging the chairs on the deck of the Titanic is the norm at the Livermore Lab, where perfectly good scientists are promoted into managers— usually bad managers. Realizing they don't know what they are doing, they hire consultants, who suggest novel ideas that don't

---

[12] P. F. Dubois, "Brain Cancer May Be The Least of Our Worries", Computers in Science and Engineering, v. 10 #3, May/Jun 2008.

[13] Warshaw, S. I., "Modeling sound fields from radially symmetric impulsive planar sources using Rayleigh's Integral", Proceedings of Meetings on Acoustics **19,** 2013**.**

work. One of my managers posted inspirational posters in the halls. Bad managers waste scientists' time, and importantly, fail to protect them from busy work from above.

Many of the units I mention in these pages were split, renamed, merged, and mangled constantly the entire 30 years that I worked there. The number of levels of management was always, to use a mathematical turn of phrase, monotonically[14] increasing.

Helping Other Programs

Since I was only half-time on Ionosonde, I had some other work in both T-Division, H-Division (Materials Science) and A-Division during these years. I helped make vectorized versions of mathematical software and applied it to Lab codes, delving into Assembly Language for the Cray-1 to add a half-precision vectorized square root, and research on a new method of radiation transport with physicists Tim Axelrod and Cliff Rhoades. In the eight years after arriving I had published many internal reports and peer-reviewed papers on a wide variety of topics.

I doubled the speed of CORONET, a workhorse application in A-Division. That meant a savings of about $27 million dollars per year in computer time. To my surprise, that year I got a very mediocre pay increase. I saw my Computations Division Leader in the hall, and I politely asked how my share of $27 million a year could be so small. He looked surprised and said, "Yes, how *did* you do that? It seems impossible."

I established a "hot line" for the part of the Numerical Mathematics Group that I now led, so that developers throughout the Lab could ask for help with our software. I took my turn manning the hot line. Quite often complaints about our mathematical software or about components I had made for the large codes turned out to be simply bugs in the customer's software. I remember two such calls quite distinctly.

A user called and complained that a certain call to a math component was not working. I asked him to describe his calculation. Then I told him, "You've forgotten to initialize the value of the variable *"i",* I said to him. There was a long pause as he looked at his code, and then I heard "Oh, yeah", and a click as he hung up.

In another case, the author of a large code called to say my new software was to blame for a terrible bug that he could not find. He was steaming mad. We talked, and I asked him how he knew the error was in my software. He said the machine had halted, and the error traced back

[14] Meaning "always"

to my software. I asked him “Where does it die?”. Then he confessed that he couldn’t answer that because “It doesn’t die if I use the debugger.”

“Are you making asynchronous memory transfers?”, I asked. Some large problems require data to be kept partly on disk or in extended memory, and asynchronous transfers are a way to do this movement in the background while other computations proceed. The only trick is to be sure the transfer is completed before you use the data. “No, no, we don’t do that.” He insisted vehemently despite my saying that he must be doing that, and that the reason everything worked in the debugger is that when a debugger stops the program, it also lets the transfers complete.

He was getting angrier by the minute, so I hurried over to look at his source code. I found the asynchronous transfer, and a search showed the code lacked the check required for its finish. “Oh, yeah,” he said. “I forgot.”

The many instances like these led some of my colleagues to jokingly refer to me as “The World’s Greatest Living Debugger.” This ability came in handy for those “situations” that continued to arise.

III. COMPUTATIONAL STEERING

STARTING A CODE FOR THE MIRROR MACHINE

The Lab was building a Magnetic Fusion "Mirror" machine as part of research into magnetic fusion energy (MFE). A Mirror machine would confine plasma in a cylinder (unlike in the doughnut-shaped "Tokamaks") and would have very large magnets at each end to reflect the plasma back upon itself. The culture of such large experiments was evolving— there were theorists, and experimentalists, as always. Now managers want to see computational predictions before building large, expensive experiments, and to guide their use.

I had completed a number of successful projects for both Theoretical Physics and the Weapons Program. Each division there had its own computer programs, and in working on all of them I had learned their idiosyncratic infrastructures. The code groups had created just a few assets, such as database managers, that could be shared between codes. The future Object-Oriented revolution would reveal that this inability to share was not entirely the scientists' fault, although the urge to invent for oneself and control everything about your code is always there. I discuss why later in "Thoughts On Software Engineering".

Among the many high positions he held, William A. (Bill) Lokke had been the Acting Division Leader in Theoretical Physics. Bill endeared himself to me as Acting Division Leader by actually making decisions. When I expressed my amazement and appreciation he said, "The operative word in Acting Division Leader is Acting."

Bill was appointed Division Leader in M-Division— Magnetic Fusion. He recruited me as the architect for the new computer program he wanted to build to model the Mirror machine. Having come from the culture of T-Division, he was shocked that Magnetic Fusion did not have the same view that simulations were an integral part of the research. They did have a number of theorists who each had their own computational models of various parts of the process.

The problem would be to glue these models together. Instead of taking some fixed input for the heating, the transport model should receive that which was produced by a heating model, which in turn should receive the magnetic-field model instead of use of a fixed assumption

about the magnetic field. The computing problem was that each process changed the input to the other processes as time progressed in the calculation. I agreed to take the position if Bill would back the vision that was brewing in my mind: a reusable architecture that let multiple codes be built upon it, reducing future startup costs.

Explaining The Terminology of Computational Steering

My idea for an M-division development system was a new concept, "Computational Steering." In order to describe what computational steering is, I have to explain a few general terms used by computer scientists.

"**Source code**" means a list of statements in some computer language. Some popular languages include *Fortran*, *C*, *C++, Python*, and many others. Each different computer chip has a language called Assembler, peculiar to that chip, that allows the very finest level of control. Above that level, the languages such as those mentioned are "high-level" languages and their statements generally look like English sentences or mathematical statements, such as "A = B + C" or "do while x > y". A large application has thousands of separate sources, to keep them to a manageable size and separate them by topic and the team members who will work to improve them.

All source code needs to be changed into something the machine can actually execute. There are two ways to do this.

1. Change the statements directly into machine code, called a "**binary**". This is called "**compiling**" the source. Groups of many binaries can then be aggregated into "binary libraries" and eventually "**loaded**" into the actual executable application. Taken together this process is called "**building**".

2. Change the statements into an intermediate form that is executed by an "engine". This is called an "**interpreter**".

In compiling a code, there is nothing available to execute until the entire set of sources has been compiled and loaded. For an "interpreted" language, the statements are executed as soon as enough of the source has been read and makes sense.

Thus, an interpreter can read one statement at a time or be used interactively. Since the execution is being done by an intermediate engine, an interpreter is usually very much slower, by at least an order of magnitude, than a compiled code.

My main innovation in scientific computing consisted of introducing a happy medium, in which "what to do" is decided in an interpreter, but actually doing it is done with compiled code, often with only a negligible loss of speed in build time. This architecture I call

"**computational steering**", and is now the preferred way to do scientific computation in many cases.

As part of the process of compiling sources, good practice is to use a "**preprocessor**". The preprocessor can either be built into a computer language itself or be a separate step that modifies the source code into an expanded text before compiling it.

A simple example is a piece of code that defines the constants *pi* and the square-root of 2, that you are using in many different source files.

If you have not used a preprocessor, then to add another constant, or to change the constants to be more accurate, you might have to do that change in hundreds of places in many different files; whereas, with the preprocessor, the compiled code gets the preferred version of the constant everywhere by your changing only the one place where it is defined, and the preprocessor copies it to where it is needed. Other examples include providing coding that needs to be different depending on the specific target computer.

I wrote the preprocessor MPPL (More Productive Programming Language) for Fortran in the 1980s. It also modernized some language constructs. It proved to be a big part of my new system.

Gluing Packages Together

Typical of most of computational science groups, the Magnetic Fusion scientists had one code that simulated one part of the physics, radial transport of heat in a plasma confined by a magnetic field. The transport code needed details about the magnetic field, so it would have a file it read to get those values. The problem was that as the plasma changed it would, in real life, modify the magnetic field. Dynamically, it was not true that the field would be static.

There were five or six similar codes covering various aspects of the machine, such as the magnetic field responding to the plasma, the heating of the plasma from external sources, and so on. Each code currently *assumed* values for the things it was not calculating, as if that package produced constant values.

By putting each code— one that did transport, one that did heating, and one that did the magnetic field, etc.— under the control of an interactive steering language, we solve this physics problem, since:

- The scientist can test each piece by itself even more thoroughly, still using static values produced by the other pieces.

- By means of the steering language, the combined packages can run a loop in which a time-step is taken with each part feeding its state to the others in turn, thus approximating the solution of the differential equations describing the total system. The language values available can also slow down the time-step or speed it up as needed.

We also begin to solve the social problem: the different authors of these "physics packages" can continue to work separately for the most part but with more realistic values from the other packages. We also unlock the creativity of the end users of the simulations, as we shall describe.

It is worth pointing out that a single scientist might be involved with a code in many ways: as the author of a package ("a developer"), as a person who uses the final product ("a user"), and as a team member or manager of the effort as a whole. Sometimes a package has several authors who must cooperate, and sometimes there are remote authors such as University collaborators. The social and technical problems presented are greatly eased by computational steering.

CREATING THE *BASIS* SYSTEM

Two important experiences inspired my design of this system, which I called **Basis.**

Professor Cleve Moler of the University of New Mexico had written a small Fortran program called *Matlab* for students to use in linear algebra courses. The original Matlab was an interactive calculator with *matrices* instead of simple numbers. All the numbers were complex numbers, and there were just a few thousand storage slots in which to store matrices.

But we had very large matrices inside our large physics applications. If there were a tool like Moler's but without the memory limitations, it could tell us valuable information about the accuracy and the speed of convergence of the algorithms used. So I took Moler's educational "toy" and modified it to use a memory manager and succeeded in using it on our data. Later Moler formed a now-famous company to make a commercial product he also called *Matlab*.

Physicist Tim Axelrod showed me how he had used the first successful personal-computer spreadsheet, *Lotus 1-2-3,* to calculate a one-dimensional supernova implosion. The physics variables were stored in a column, and every time Tim touched the recalculate button, the macros he had written took a time-step from the latest column and wrote the new values into the next column. He could even graph the results. This was profound: the people who wrote

“1-2-3”, I thought, had no idea you could do such a calculation, and certainly not the knowledge of how to do it, or even that such a calculation would be useful.

Developing my steering system

I had the idea to create an interactive programming language resembling a version of Fortran in which the first-class objects included vectors and matrices in addition to scalars, strings, and characters. It would superficially look like Fortran with all the features of a real language: if-tests, loops, functions, subroutines, etc. To this I would add the ability to access the variables that appeared in the physics code. If there were a two-dimensional real array named “heat”, a variable named “heat” would be available in the language.

The user would be able to run the simulation to a point and then examine, calculate with, plot, or change the contents of “heat”. Since another physics module might have a different variable also named “heat”, I designed a “package” architecture having “namespaces” implemented as a “dot” notation to distinguish the package to which the variable belongs. This would allow physicists to retain control over their own work, and each could develop their programs in isolation they way they were used to doing. I viewed this as an important sociological feature of the architecture.

The “language interpreter” would become the input mechanism for the user. I wanted it to look superficially like an array Fortran because that would seem intuitive to them. Fortran was the only language most of them knew. I needed to learn how to write a language interpreter, and figure out a way to make it easy to get the information about the physics variables into my “run-time database” without requiring the authors to do a rewrite of an existing code. After stripping out an existing user interface and a time-step structure, I would apply tools that would write the “glue code” to connect the physics packages to the interpreter. I would supply an interface to other assets such as databases, math modules, and graphics.

The social part was never going to be easy; none of the developers with their own packages was going to want to do anything new, but with the backing of Lokke, I knew it was possible. The “hook” would be that they could develop their packages with less effort in less time and get more done scientifically than before.

Starting the MFE Code: “MERTH”

I arrived in M-Division in 1983 and was assigned Peter Willmann to assist me as my programmer. Peter turned out to be a great friend, and he was the bravest person I ever knew. He was a small person, born with *osteogenesis imperfecta,* which meant that his bones were

fragile. He had been kept in bed until he was 12 but told me he had still broken bones a hundred times. He had, however, gone to Harvard. Wheel-chair bound, he was one of the best "birders" in the world. When he went on vacation, a very large friend would go with him and get him up trails such as those in Papua, New Guinea. He eventually retired to San Diego and died of old age. For a person who had had to bribe machine-gun carrying terrorists to get to see some bird in Senegambia, living to die of old age was, like the rest of his life, a miracle.

Peter was very passionate about the new project. He had worked on a graphics library already that we could use. The code that Basis would build for MFE to model the Mirror machine was named MERTH: Magnetic Energy Radial Transport and Heating.

By the summer of 1984, we had the first stand-alone *Basis* running attached to a graphics package but with no physics. I had designed a tool that took the source for the Fortran common blocks and spit out an "include" file to incorporate into the source code. It also produced input for Basis' run-time database to allow the interpreter to "see" those variables. I used my preprocessor MPPL to add features to Livermore's version of the Fortran compiler to make it easier to write the physics and for macro expansion. Eventually Fortran 95 would resemble it strongly. MPPL took the common file and the physics sources and produced standard Fortran to be compiled.

Very quickly I had two of the most important MERTH physics components "talking" to one another. The ability to run parameter sweeps (running the program repeatedly with the same input but with just one input variable changing) led to the discovery of a bug that had been in one developer's package all along. The physicist said the parameter sweep and graphing capability had let him do in an hour what he had been spending days doing. Word spread, and enthusiasm grew. People started converting their other program to use Basis.

Writing tools that manipulated mostly text was not simple in Fortran. I had to research how to write a "language parser" and a run-time database in Fortran. I wanted to use the C language and other tools that were a part of Unix. I ended up finding a PDP machine with a C compiler for these tools, but I couldn't have these be part of the Basis product because when I requested that a C compiler be installed on the MFE Computer Center machines, they refused, scoffing at a project needing anything but Fortran.

By 1987, we had a very successful system within Magnetic Fusion Energy. MERTH had integrated the major components that had been planned for it within the year deadline Bill

had given me. We wrote a users' manual[15] that listed a large group of people who had contributed to Basis, including Peter Willmann, lead programmer; Cathleen Benedetti, a student from U. C. Davis/Livermore, Mark Durst of the new Computer and Math Research Division, Andrew Zachary, a post-doc, and David Roberts, a "Superkids" summer intern. The manual was co-authored by Professor of Computer Science Zane Motteler, a summer faculty from Cal Poly; later Zane became a full-time employee and Basis maintainer after 1999. Special mention goes to computational physicists W. M. (Bill) Nevins, Ron Litterst, Alex Friedman, Lynda L. LoDestro, Max Fenstermacher, Gary Porter, Marvin Rensink, Tom B. Kaiser, Bruce I. Cohen, and Gary Smith. The excitement from this group is what spread our ideas, along with M-Division leader Bill Lokke's unwavering support and encouragement.

Spreading The Idea of Basis

My architecture for scientific programs, an interpreter language as a user interface to direct the computation that uses packages of compiled routines for speed, is now called Computational Steering. The group I worked with later, the *Kull* Project, uses a Python interpreter over compiled code in both Fortran and C++. Doug Miller of that group tells me that their new hires now assume computational steering. "Of course, how else would you do it?", one said.

That was very much not the case at the start of Basis. Quite often I sold the idea by doing the conversion for the physicist. It took just a few hours per code, since I was mostly deleting the user input and the main loop. My collection of automated tools made this quick and easy. Then I showed them what they could do now that they could not do before. Everyone was happy and amazed by the new capabilities. Nobody ever went back after trying Basis.

I stopped counting after hearing of 200 conversions to Basis— small efforts by one or two people, and large efforts. I interacted with General Atomics, Oak Ridge National Laboratory, Los Alamos National Laboratory, and the larger MFE community. We even had users'-group meetings[16]. In the end, the idea of using an interpreted language to steer compiled assets won out. I was gratified when Travis Oliphant, the creator of *SciPy*, took the time to notify me, more than a decade after my retirement, that a Python-steered data-analysis tool had been instrumental in the discovery of the gravity waves Einstein had predicted.

---

[15] Dubois, Paul F and Zane C. Motteler, BASIS User's Manual, Livermore, Calif. Lawrence Livermore National Laboratory, M-189, June 9, 1987.

[16] P. F. Dubois, et al., Proceedings of the First Basis User's Group Meeting, Lawrence Livermore National Laboratory, Livermore, CA, M-4690 (1988).

* * *

Steering Enables Teamwork

A major quality of steering is that it greatly reduces the need to modify the compiled part of the program, which is where the difficulty in group work chiefly lies.

Prior to steering, a user, let's call him Ralph, might need to add a "diagnostic", a small piece of code that produces some quantity computed from the state variables. This diagnostic may not be wanted long-term or by other users, so it also needs a "switch" in the user interface to request that it be done.

Without steering, user Ralph would have to ask a code author "Judy" to add the diagnostic. If the need was urgent, Judy would not only modify the current public code to make a private version for the user, but also duplicate this work into her own current work being prepared for release. Ralph would be "stuck", perhaps for months, using that base release with the diagnostic added, missing out on any improvements or bug fixes that were added elsewhere, until the diagnostic made it into a major release. Perhaps, two years later, if you asked code leader "Phil" about this strange code piece you found, Phil would figure out that Judy added it, ask Judy about the diagnostic code, and she would be apt to mutter, "Oh, that was for Ralph, and he doesn't need it any more," creating another job for someone. Likely that somebody is, in practice no one, and the debris pile accumulates. I liken it to hardening of the arteries, because in five years no one knows what it is or why it is there, and so they do not dare remove it or the user input switch that came with it.

With steering, Ralph could simply change his Basis input to execute the desired code after every time-step and add the diagnostic to the output dump. This scenario was so common that later I wrote a "history package" that any code could use to generate and output diagnostics. Using Basis, the diagnostic was never added to the compiled code.

Similar remarks apply to other calculations wanted by only some users. For example, if energy is being deposited in a plasma from an external source, the numerical mesh may over-concentrate the heat deposition. One user had such a problem and simply used Basis coding after the heat was deposited, diffusing it somewhat before continuing. Since Basis is an array language, the run time to do this calculation was only slightly more than it would have been to compile it. There was no need to add this physics to the compiled sources and no need to add a user interface to it.

Importantly, this applies to the program authors as well as the program users. When a user runs a Basis application, he or she does not need to know or care whether an author had included certain physics at the interpreter level or at the compiled level. This allows program

authors to do parts of their work in the interpreter if that calculation is not expensive, or to prototype new calculations quickly before doing the laborious task of writing them in compiled code.

Steering Enables Scientists' Creativity

When you have scientists as users, they think of many clever ways to use steering. For example, they could intervene in the run periodically to make a frame of a movie, modify some values, coordinate with another package, and so on.

Perhaps the most spectacular instance of this was a project physicist Judy Harte did. She prototyped a program to calculate radiation deposition in human tissues. She computed doses for radiation therapy of the lungs or the prostate gland that were much more accurate and therefore did not degrade bones. She did this by accessing the fast compiled physics routines of an existing code built using Basis, and produced the rest using the Basis interpreter. With this proof of principle, a commercial startup wrote a separate, unclassified, stand-alone program which they called "Peregrine". No one would have ever gotten funding to build such a prototype on speculation because they would have had to reproduce all the complicated compiled code just to try the idea.

Being aware of this potential, we see that the user-interface language must be clean and readable, and easily learned at least to the level needed to operate the physics application. If the language has extra power hidden under the hood, so much the better. Having made the Basis language very similar to Fortran was one key to its success, since the scientists already knew that language. Once a scientist was given a Basis program, their creativity sold it to others.

Flow of Execution of a Steered Application

Application contains the interpreter for the user input language (e.g. Basis)
At startup, Application executes the "glue" to enable access to compiled assets
Next, Application interprets parts of the code the authors wrote in the input language.
Then, Application interprets user's input written in the input language.
When interpreting, Application uses the "glue-code" when needed to execute compiled code.

Resistance

Computational steering was completely novel. There was no good metaphor for it. "So it is like subroutines!" one person said. I explained it over and over, but lacking an X in "So it is

like X", I often found it hard slogging. While there are legitimate concerns about adopting new tools, as I discuss in the section "Thoughts On Software Engineering", more often what I encountered was just a closed mind. Here are three examples of the resistance I met.

Resistance: Two Files Is Too Many

Bill Lokke asked me to see a scientist in MFE who had a "one-file" code to see if he could convert it to Basis. In a "one-file" code, the source is just one file, either including the input data as source or it contains code to read the input data from a different file known as an "input deck". Shared data areas ("common blocks") are simply copied and pasted where needed. This remained a common practice for lone practitioners for the rest of the century.

I started to explain that the first step to use Basis is to take all the common block statements out into a separate file in my format called a "variable descriptor file". Eventually I had an automated tool to do that for you. The appearance did not change much, and you could add documentation about variables and functions that would become available to the code's end user. The information gets into the source code by use of an include-type macro. The Basis system arranges for all that to be available to the end user.

I got no farther than "separate file". "Two files!", the scientist cried. "That's just unacceptable! The whole code has to be in one file!" That sort of response is common when new procedures are introduced; the person who is considering someone else's idea is awaiting the first thing they can object to in order to shut the idea down.

Resistance: Tokamak Systems Try to Get Together

I was asked to help with a project to build a Tokamak "system code" from collaborators all over the country. The collaborators would develop their piece and send it to a team at Oak Ridge National Laboratory who would assemble the pieces into a program for simulating Tokamak systems. They had tried to do this once already and failed.

One group argued that the "problem" had been inadequate comments in the pieces so that the "glue" people made too many mistakes. They wanted to try it again with more comments. The Livermore Lab's representative to the project had become an ardent Basis fan and converted his code to it. So he asked me to participate in a system code meeting and show them Basis.

I gave them my Basis talk but one loud and very Angry Guy, a famous and politically powerful Professor from the University of Chicago, insisted that this was all nonsense, that all they needed was more comments, and in the end the Angry Guy prevailed.

Six months later, I was told to go to Oak Ridge to see if I could salvage the new code, which was failing again in exactly the way I predicted. I spent two weeks there. We converted each of the contributions to run under Basis so that we could at least test the pieces. Sure enough, the submission of the Angry Guy worked if you ran the one test he had set up, but failed if you called the subroutine twice, and blew up if you called it with almost any different data than in his one test.

Soon we had the combined code running, successfully. We sent back the Angry Guy his subroutine with the corrections I had made to his very old-fashioned and elementary programming, explaining that it now got the right answers. Angry Guy went ballistic: how dare we touch his code? Angry Guy intimidated the management, so the Oak Ridge boys had to glue the pieces together without Basis, but they could do that by secretly keeping the Basis version running to check against, as they told me. I got a weekend at Dollywood out of it.

Resistance: Suivez L'Argent

Princeton had political control of much of Magnetic Fusion due, I believe, to their proximity to the East Coast centers of power. I gave a talk at an MFE meeting about Computational Steering after I had converted Livermore's LASNEX in the mid-90s, at the request of the Livermore team, but one Princeton scientist at the meeting named Steve ranted that he had seen the (classified) source code for LASNEX, and in his opinion it was terrible. We said nothing because most people knew that Steve was either lying, or he'd committed a felony, the LASNEX physics source being classified. Steve wanted the money for Princeton.

IV. LASNEX, ICF'S PRINCIPAL PROGRAM

MODERNIZING LASNEX

Livermore Lab's flagship program for Inertial Confinement Fusion[17] is called LASNEX. It is now fifty years old, which seems unimaginable for a computer simulation. It was founded by George Zimmerman who came to the Laboratory in 1969. Among his accolades, Zimmerman received the 1983 E. O. Lawrence Award for contributions to national security, and the 1997 Edward Teller Award. He was certainly unbeatable as a supervisor.

The LASNEX computer-science team was asked to evaluate Basis as a possible new improvement to its development system and user interface. The code was written in LRLTRAN, Livermore Lab's local dialect of Fortran, and it used a lot of "macros" (shortcut words that expanded into code). The LASNEX team had their development system and code environment and were working on a new user interface, which they were about to release.

The team doing the Basis evaluation did not, of course, want to be told to use Basis, as they would rather continue as they were. So they cited various problems they thought would rule out using Basis. The analysis was shallow. I didn't bother to argue, as this was what one would expect.

Alas, when they did their own thing, it wasn't good. Users were complaining. The computer-science team leader quit. I was asked if I would consider coming over from M-Division. I said no twice. But then, the Department of Energy canceled the Mirror machine. DOE had decided it could not afford both Tokamak and Mirror Research. MFE would turn all its energy to Tokamaks. I felt that while I could certainly stay in M-Division, the support would slowly decline as Princeton's political power gave it the advantage in the battle for scarce resources. When physicist David Kershaw, the LASNEX group leader, asked me again, I told him I'd come listen.

I arrived for the interview in Kershaw's office, and almost immediately another physicist I knew well, Bruce Langdon, barged in to give me a list of what he needed fixed right away.

[17] Also called "laser fusion" to distinguish it from "magnetic fusion".

David and I laughed. “You know this is an interview, right?” Kershaw asked Langdon. Customers needing emergency corrections was not a good sign. Kershaw, however, was very open to my ideas, and he outlined the big problems he faced:

- The Lab was phasing out having its own compiler and operating systems groups, and the program would need to be converted to use standard software supplied by the computer manufacturers.
- It took an average of three days to entirely recompile and reload LASNEX. An elaborate system for partial rebuilds was clumsy. Refactoring (restructuring) the code was very difficult because no author could afford the time-cost of the full build, so the code got crufty[18].
- The source-code control consisted of remembering a base version, and “diff” files for each subsequent change. The diff files were used by an editor to update a step at a time to the current version.
- The new interface was not working and needed to be fixed or abandoned.
- Kershaw believed that the code needed to run on the new, powerful “workstations” such as those being produced by HP and Sun, in order to have enough computer time for the physicists to do all the calculations that would be needed for a national priority, the Comprehensive Nuclear Test Ban Treaty[19]. The U. S. couldn’t agree to its terms until we were able to reliably replace testing our stockpile with results from simulations.

I decided to take the job. After arrival, I discovered there was not enough testing of the major components, or of the complete code. Each developer had a collection of test-problems to verify their own work, but there was just a single test problem for the integrated system for an update of what we called the “experimental version”, and a more elaborate set of problems reserved for major releases which happened only a few times a year.

As I had promised, I dug into their new LASNEX user interface to see if I could fix it. I found the architecture of it to be just wrong; someone had winged it without reading the literature on compilers, as I had done. And, just fixing the interface was going to eat up resources without dealing with any of the other goals. It was a choice between trying to deal with each of Kershaw’s problems one by one, or to set out overland, converting to Basis. To

[18] Cruft(n): Badly designed, unnecessarily complicated, or unwanted code or software. — Oxford Dictionary

[19] Signed by the United States but the Senate refused to confirm it. Nevertheless, it is followed.

do so would mean nothing much would be working until it was all working. A Wall Street prospectus would call this “management panics risk”.

The task was formidable. I wrote a one-time tool to use to transform the source code into Basis’ pre-processor MPPL, removing old macros and replacing them with equivalent coding that could be translated into standard Fortran. The tool extracted the common-block information into Basis’ variable-descriptor files. The idea was for each physicist to use the tool once on their own package sources, then clean up what remained by hand. The Division bought a farm of HP Workstations, and I and the team got everything running. The source to be compiled was now standard and could run everywhere.

The users loved it. The developers loved it. Gradually both developers and users came up with very creative ideas and shared them with each other. The testing regime and source-code control system I introduced greatly enhanced successful teamwork. The days of a developer adding bad code to the repository were virtually over. I eventually reduced the “build from scratch” time on the supercomputers from three days to a couple of hours, reducing the barrier to code refactoring to something bearable.

I had to manage the politics of implementing the change as well. I visited the person in charge of CORONET, another workhorse code in A-Division, near the beginning of the project. He said the whole project was foolish, that I would never be able to get a big code running on workstations, and that he doubted the users would accept the new interface.

There was domestic harassment too. Dave Munro was a physicist who had created his own “post-processing” tool, an interactive tool named “Yorick”. As it happened, part of what I had to do also produced a by-product Basis code that could do the same things. Inevitably, factions formed as to which was the best one. I thought Yorick a clever effort and didn’t care what people used. Yorick was a stand-alone program and not a rival as an interface to LASNEX itself. I helped Dave publicize Yorick in my column in *Computers in Physics*.

We were told that the added capacity made a crucial difference in enabling the United States to sign the Comprehensive Nuclear Test Ban Treaty of 1995. The Treaty was adopted by the U. N. in 1996. In 1991, the team received the Physics Distinguished Achievement Award from the Department of Energy for “Work-Station LASNEX”.

I was with the LASNEX group for a decade. After I left, a variety of team members took care of Basis. Zane Motteler[20] retired, and others took over in turn, one being Dave Munro, who of course was the most qualified.

Pioneering "Work From Home"

During my years on LASNEX, my personal situation got difficult. Amongst other problems, our son had learning disabilities, which later proved to be autism. There was a lot of testing, interacting with schools, therapies, etc. As it happened, this was also a period where computer scientists were in demand in other enterprises. I would get frequent inquiries as to my availability for a position.

One inquiry was specifically because I was known in the Eiffel community. I talked to the company CEO on the phone and learned it was a telecom-related company— something to allow cell-phone communications inside big conventions. They wanted to program the switches in Eiffel. However, the company was located in Reno, as taxes and living costs were lower in Nevada. Another company I'd visited made Fortran compilers for small computers; they had their offices on the East side of Lake Tahoe in a little ski town, for both tax and recreational reasons.

I was never going to be willing to go to Reno, but the CEO said that I could just work at home, maybe fly up once a month for a couple of days. I met with them in person and didn't like the vibe, but found I was really tempted by the work-at-home part.

I mentioned this temptation to Judy Harte, one of the LASNEX physicists. She said, "Well, don't take the job just for working at home without asking if they will let you do that here. I bet they would." And indeed, my management would. Although I was leading the group of computer scientists attached to LASNEX, they were willing to give it a try.

A key point was that it was now possible to have a new kind of home phone connection called a "Digital Subscriber Line." That gave me the computer power I needed. And, it was possible to arrange to forward my Lab phone to make it ring at my home. That way, people did not focus on whether I was "here" or "there". My wife set up a desk in the living room, and after I heard some drilling and hammering sounds, an ethernet cable dropped down out of the ceiling, having run from the one place I could attach the DSL, over two bedrooms into

[20] Motteler had decided to leave Cal Poly and come to work for me when the State of California stopped paying the professors during some budget fight. The Lab dodged the problem by saying yes, we are managed by a state agency but all the money is Federal.

a bathroom, and down to the ground floor. A woman willing to have an ethernet cable dropping down into an office in the middle of the living room— her price is far above rubies.

After a year, George Zimmerman said he thought that working at home had not diminished my contributions in any way, and he'd received no complaints about it. When people had to start working at home in 2020, I knew they would love it. It fixes what is wrong with Western Civilization, namely commutes and long absence from family. I continued to work 60-80% of the time at home for the next ten years. I was able to work from home between 40-80% until I retired.

V. PREPARING FOR THE FUTURE

"The problem with top-down programming is that real applications have no top"— Bertrand Meyer

OBJECT-ORIENTED PROGRAMMING

The big fad in programming had been "top-down programming". Common wisdom suggested that you view the program as a task, and break it into smaller and smaller subtasks, until the subtasks were easy enough to code up. The problem was, as Meyer pointed out, the scheme is based on a false premise: that we know for certain what the task is, and are certain it will never change.

I began to hear about three separate efforts to create new programming languages based on a concept called Object-Oriented Programming (OOP). The idea was to form the program around "objects" rather than "subroutines". This would require the scientist/programmer to model the concepts in the program rather than the work-flow, or sequence of operations. The objects a program is talking about change much less than whatever our current tasks are. We are actually programming bottom-up in a sense. Writing the program based on communication between relatively stable objects makes the program reliable, enables the reuse of components, and makes the program easier to modify. The last few lectures I gave before I retired highlighted the high rate of change in scientific programs, since a program that isn't changing is probably near its end of life. In one "quiet" year, LASNEX had 75 sets of changes to its main-line repository.

Objective-C, *C++*, and *Eiffel* were all new OOP languages. The first two were extensions of *C*, which had taken over as the principal language of non-scientific programming. I attended a conference about C++ in Santa Fe, and I saw incredible excitement. Bjarne Stroustrop, the inventor of C++, was being followed to the door of the men's room by a ball of people wanting to talk to him.

I also attended a one-day seminar on Objective-C, but it looked unsuitable for me as it would not be available for supercomputers. It had been developed as the language for Apple to

write macOS, iOS, and their Applications Programming Interfaces. Objective C was replaced by *Swift* in 2014, and I have used it in the Apple Development Environment to write a home-hobby program, "Bridge Clock."

Eiffel,[21] by contrast, was a completely new language designed by Dr. Bertrand Meyer, one of the authentic geniuses of the computing revolution and my principal mentor in this new way of thinking. Meyer had recently left a faculty position at U. C. Santa Barbara to form *Interactive Software Engineering*. His abstract goal was to be able to prove that a program is correct. His practical method, *Design by Contract*, states in the source text-file a set of properties that the objects were to have before and after each operation, and the set of properties expected by the "methods" on the object that carried out changes or revealed information about the object. I attended what turned out to be his first educational seminar at the new company.

I could not use Eiffel on a supercomputer, but I did learn it on my home computer. Physicist Cliff Rhoades, an influential voice in computational physics at Livermore, took up my enthusiasm. We tried to get money for a pilot project but were unsuccessful, as decisions are made mostly by the physics staff who may not be aware of the benefits of new computational developments. I'd had quite a time selling Basis as an abstract idea— it was only when physicists used it that they had that "Aha" moment.

I became the leading advocate for Object-Oriented Programming at Livermore, along with Cliff Rhoades. I taught classes in a series I humorously called "Dubois Free University," which eventually led a group of the younger physicists to start using *Python* and C++ for large, multi-author projects, as we will describe.

A few years later, I got a call from the Associate Director for Computations, Bob Borchers. He wanted me to come to a meeting to present the Lab's efforts in OOP to some visiting Hewlett-Packard executives. In a Dilbert moment, I said something like, "You know we haven't got any such efforts, right?"— resisting the urge to add, "You refused to fund any of them." But he insisted, and I gave my short talk to a conference-table full of HP executives on OOP's benefits and why we were interested. One of the executives said, "We're doing that; obviously it is the right thing." Then he turned to Borchers and asked why we weren't doing anything. I could have kissed the man. But nothing changed in Computations, adding more evidence to my idea that the *Dilbert* comic is a documentary, not a work of fiction.

[21] Meyer, B., *Object-Oriented Software Construction*, Prentice-Hall, 1994; 2nd Edition, 2000.

Knowledge Without Experience Usually Fails

There were, of course, computer scientists with different ideas for new languages. Like Communism, these ideas are often based on an intellectual idea that seems attractive on paper but doesn't take into account how real people work together. One useless but repeatedly-funded Lab effort consisted of a functional language, which is almost the opposite of OOP.

The problem with language design is that you can write a few-thousand-line program in almost anything and not see any problems. The problems occur at scale and as codes age. It would be better if the computer scientist first had to be an applications programmer, so that if not "down with the struggle" he or she would at least know what the struggle is.

An early language was *Forth*, which was, if I remember correctly, a reverse-Polish notation language such as those used on the early HP scientific calculators, and a big favorite amongst telescope operators.

To try Forth, Bill Lokke and I bought, for home use, new Atari 800 computers with floppy drives, tape backups, a modem, and "game" cartridges for Basic and Assembler languages. We each spent about $2600 on these. One Christmas, Bill surprised me with a floppy and a printout of a one-D implosion program he'd written in Forth, which I gave to the Lab archivist when I retired.

None of these odd languages are viable in real life, especially for large teams of scientists. The closer the author is to academia the worse the result is likely to be. I remember visiting Zane Motteler when he was still at Cal Poly. Zane knew I was implementing the Basis Language, so he arranged for me to meet the faculty member who was his department's "Compiler Expert". This nice gentleman informed me sheepishly that he'd never written an entire compiler.[22] Unfortunately, he knew nothing of use to me.

PYTHON COMES TO THE RESCUE

"Simple is better than complex"— Tim Peters, *The Zen of Python*

I could see the writing on the wall: Object-oriented programming was going to take over, and people would want to write scientific programs in languages other than Fortran or in a

[22] Al Shannon and John Engle of our compiler group had guided my reading of Aho and Ullman's *Principles of Compiler Design* (1977) and its follow-on from 1986. Together these were called the Green Dragon and the Red Dragon, named after the denizens drawn on the covers.

mixture. But how could Basis steer an object-oriented language? Basis' interpreted Fortran-like language would not be enough to express the programming concepts in the scientific modules that would be written in these new languages.

I began designing the new language as a research project, while my team and I were keeping the wheels turning on LASNEX. Then I happened across a description of *Python,* an open-sourced language. Python was an interpreter like Basis but had OOP at its heart. Most importantly, we could extend the language by writing C modules following a certain protocol and thus connect pieces written in compiled languages to the Python interpreter. Crucially, the Python language itself, which the scientific users would use for input, was simple and straightforward, not a computer-sciencey mess full of braces and semicolons. I immediately decided that all Python needed added to it was a facility for processing numeric arrays at high speed, and then it might meet our needs.

Numerical Arrays Needed For Python

A small number of people around the world were interested in data processing using Python, and with our new ability to communicate on the Internet, we started discussing design in an interest group called "matrix-sig". (A "sig" meant "special-interest group").

The author of Python, Guido van Rossum of the Netherlands, was open to expanding the syntax of the language if we could agree on the right design for the Numerical Python addition. The fact of Python being open-source, with Guido retaining artistic control but encouraging contributions, was the key to Python's success— it is today one of the principal computer languages.

Our first success was agreeing on a notation for complex numbers. Should the imaginary "i" be written 1i, or 1j? Different communities in science used one or the other. My math side, 1i, lost out, and the engineering 1j was added to Python.

For array operations we needed to blend in with the already-existing subscript notation x[i]. We added the idea of multiple subscripting x[i,j] to the existing x[i][j]; such access would be necessary for fast operations. Once we had the basic design, one member of the group, a grad student at MIT, James Hugunin, wrote the "Numeric" extension. It was an immediate success. Hugunin left MIT a year later, and I took over as the "Head Nummie", coordinating the Numerical Python extension group. Running an open-source software project was difficult, since the Lab had intellectual property people to wrestle with. Years later, we succeeded in making mainstream this new sort of cooperation with the world. I suffered the

usual arrows in the face of any explorer[23], and I carried on in this capacity for five more years. Eventually the different interests of different communities resulted in a bifurcation of the Numeric effort for several years.

Then, a hero arose, computer scientist Travis Oliphant in Utah, who made a new version of Numerical Python with enhancements that satisfied both Numeric communities. He actually formed a company around supporting users of his "*SciPy*" and has been very successful. This third-generation Numerical Python has taken over the world of scientific computing.

I wanted to get started modifying the Basis system to combine compiled assests with Python, but became frustrated with my management, who felt we didn't have the resources to do it and also did not understand why we should. I'm sure they thought that Fortran would continue to be the only language scientists used. In the end, a few years after I retired, they did have someone do that modification.

Python also proved to be a much better language for writing some of the tools that were part of the Basis environment. I was able to rewrite tools to make them much more maintainable by others. I was spurred to do this when Jim Crotinger, who holds a Ph.D. in Physics from MIT, told me he couldn't figure out how to modify one of my tools written in the Unix tool *Perl*. If you need more than a doctorate in physics from MIT to work on something, it is too hard!

Python is now used everywhere in scientific computing. The community kept it free, the world added virtually all the math and science modules you could imagine, and eventually, as I will describe, Livermore Lab used Python to steer C++ and Fortran components.

[23] Rice, Edward, *Sir Richard Francis Burton,* N. Y., Scribner's, c. 1990. Burton, an explorer, took an arrow in the face soon after landing on a beach in Africa. He pulled it out and kept going.

VI. THE ARTIST'S BLUE PERIOD

SECURITY UP, SCIENCE DOWN

Wen Ho Lee was a mechanical engineer at Los Alamos. He was working in a classified position very similar to mine in a code group, although I never met him. Lee was arrested for moving his classified source code to an unclassified system, a class-A security violation.[24] Much legal wrangling and FBI investigation ensued.

News reports did not mention that Lee's behavior included other actions that had brought him to the attention of the FBI. We learned later that he had been given four polygraphs; he passed the first one, and then they told him he failed the last three, and the FBI retroactively decided he'd failed the first one. DOE retroactively increased the classification level of the material he had moved to make Lee's action criminal.

Thus it appeared to me that what he had done did not merit the consequences he suffered. He was held for nine months *in isolation without bail and constantly shackled.* Eventually he was acquitted of all but 1 of the 59 charges against him; the judge apologized to him, sentenced him to time served, and the government paid him $1.6 million in damages.[25]

I was now based in Bldg. 111, the main administration building. As a part of the security, the elevator now had slots into which you inserted your badge to gain entry to certain floors. The rule was to challenge anyone you did not know who rode with you to your classified floor and got out. I understood, but in real life, challenging someone had its drawbacks. When we were told these rules, we joked about yelling, "Up against the wall, <expletive>!"[26] to an administrator or visitor.

In another security event, an employee at Los Alamos lost some unknown material that later turned up behind a copy machine. The security screws were tightened further. Now we had

---

[24] If a computer program is classified, all the source code, including the unclassified portions, is by default, without review, classified "Protect As Restricted Data" (PARD).
[25] Hillary Clinton deliberately mishandled classified material as Secretary of State in an egregious manner without consequences. The press treated it as a mere partisan matter. This made me furious.
[26] A phrase used at the demonstrations in Chicago, 1968

to lock our offices with our secure keys even if we went to the bathroom and back even if we had no classified material in use. Was there a spy behind every bush?

Spooked, the DOE started insisting on periodic polygraph tests for program personnel. There was general opposition and repugnance. Scientists all know how unreliable polygraphs are; evidence from them is not admissible in court. We heard that some actual spies pass such tests— apparently between drugs and training this is not difficult. The worst part to me was that the security people said they believed the tests are almost always right. Some Lab managers pushed back hard, but some key personnel were forced to comply or quit. I planned to move out of the program rather than put me and my family at risk of a false accusation from a flaky procedure.

With me on edge, Physicist Doug Post[27] arrived to supervise all the code groups in A-Division. One of the first things he wanted to do was send out a message to everyone. There were two computer networks, a classified one, and an unclassified one. The wiring for each was kept a certain distance apart to avoid induction effects. Post wanted to send out an unclassified message, but at a meeting he said he did not know how to do this yet, so I casually volunteered to forward the message if he just sent me an email. I received Post's message and forwarded it without reading it.

The next day someone came around to inform me that I might be in big trouble because that message had classified material in it. He said everyone knew that I had only forwarded it but that I was responsible anyway. Technicians were to track down the recipients and wipe the message from all the servers and hard disks, including any personal home computers. They informed me that I "probably" would not be fired. A few days later authorities decided the message was not classified after all.

Doug Post became a fan of my work. He loved my ideas about software quality. Excited, he started pushing me to be on a DOE committee that would tour DOE facilities and develop some guidelines for software quality. I viewed the task as certain to fail and said so. I might develop a standard all right (and I could do that sitting at my desk), but my experience told me that getting anyone to follow it was another thing. We all made jokes about the International Standards Organization standard "ISO 9000" used in other enterprises and portrayed in *Dilbert* as only useful in creating binders full of paper. I said that I would not go on the tour.

The straw that broke the camel's back was a false "whistle-blower" complaint. I was asked to come to the Security office, and I had an interview. The complaint said that I must be getting

[27] Not to be confused with a Dick Post I had worked with, the famous Magnetic Fusion leader.

kickbacks from Bertrand Meyer's company for sales of his Eiffel software to the Lab. The detective interviewing me agreed immediately that the problem with that theory was that there hadn't been any sales of Eiffel software to the Lab. There was also mention of my presentation in Barcelona. Meyer had been invited to give a keynote address to an International Geophysical Union (IGU) meeting in Barcelona, but discovered he had a schedule conflict. He suggested me as an "object-oriented scientist" to give the talk. IGU invited me, and my supervisor approved the trip. As I discussed things with the detective, I learned that he had interviewed my supervisor, George Zimmerman, who told him he thought my interest in the Eiffel language was entirely genuine. I had written *EiffelMath*, and Prentice-Hall published a book about it, entirely at home on my own time with my own computer.

As the interview wore down, the detective asked if I had any thoughts about why someone would file this complaint. The whistle-blower laws violate the right to confront your accuser — someone can make the complaint anonymously and receive a payment if wrongdoing is uncovered. My former colleague Mary Willard from Canada had told me of someone in Colorado who was apparently making similar accusations to try to get a payday, and I had met this person at a meeting.

"Sometimes when you are successful, people are jealous," I said. The detective nodded his head yes vigorously, stood up and shook my hand, and said that this issue was over. "Watch your back over there," he said pointedly.

I felt that the atmosphere had become too frightening, and my chances of getting to do the research I wanted were too low with the current bean-counting. I was excited by computing on smaller machines too, and I looked for a chance to do that.

I Move To Climate Modeling

I saw an opportunity to work on an unclassified workstation and PC-based project, creating the software for analyzing and comparing the output of numerous climate models around the world, in the Program for Climate Model Diagnosis and Intercomparison (PCMDI). I went to an interview in the middle of the day, just after Doug Post told me he was going to assign me officially to the DOE software standards panel and said I was just going to have to be on it.

PCMDI was of course glad to get one of the most well-known computational scientists at the Lab— I got the job on the spot. After I got back to my office, Post stopped in to see me. I told him I had thought it over, and that I had decided to quit instead. You should have seen his face! He was a nice guy, and I later explained to him that the issues for me were the working conditions and the ability to work on something new.

I requested that they cancel my security clearance, but my request was refused. They said we could revisit the question in a few years when my renewal would be due, but said they wanted it kept for now because it would be so expensive to redo it later if needed.

At least I didn't have to be on that DOE panel. I never heard any more about it.

I UNLEASH MY INNER WRITER

When I started feeling discontented about my work, I went to the Lab's counseling department, and they had some vocational tests I could take. When the counselor went over the results with me, the schism at the heart of my education was apparent.

"You should have been a history or English professor, or a writer," according to the tests. But the counselor went on to say, "Your job pays too much to change now. " This echoed my father, who twenty years before had said, "Math pays more" in suggesting I major in Math, not English. My mortgage seconded the motion. Fortunately, the counselor had a suggestion: make Literature or Writing an avocation.

Beginning Thirteen Years As An Editor

Less than a month later Lewis Holmes from the American Institute of Physics called to ask if I would like to edit a column on scientific computing for the bi-monthly magazine, Computers in Physics. Armed with my counselor's advice, I said yes, but added that it would be important to me not to simply edit the contributions of others but to write a "sidebar" for every issue. They agreed.

I loved being an editor, and I think I helped many authors improve their papers. I wrote a few myself, and the side-bar was great fun. Later the magazine expanded into *Computing in Science & Engineering*, and the "Scientific Programming" department became one of their most popular features. The side-bar became "Café Dubois", with a cartoon of me sitting at a French café waiting for you to come sit and *discute*.

* * *

Predictions of What 2007 Would Be Like

In 1997 the department editors of *Computers in Physics* had a round-table discussion for an issue about the future, and about what all the editors thought 2007 would be like. As usual, I endeared myself to the discussion by being a little over-the-top with ideas, so they decorated the article with a cartoon showing me thinking of these ideas. Illustrated clockwise, the cartoon shows the dog-years represented the way change now seemed to be accelerated. Remote learning, custom clothing manufacture, the internet of things, producing and printing written things locally (PDF, e-pub) and personal journaling on devices have all come true by now.

Café Dubois was voted the most popular feature in *CiSE,* and one year they made a disk of

all the columns up to that point as a subscription-renewal incentive. You are only as good as your editor, so equal credit goes to *IEEE* editor Jennifer Stout Ferrero. I wrote for *CiSE* until 2008[28].

I also became very interested in Victorian Literature, and Charles Dickens in particular. U. C. Santa Cruz has an annual summer conference called "The Dickens Universe" and I have attended six of them on my vacation. It is an interesting mix of amateurs, who've actually read all the books, and professionals, who haven't but who have all the theories.

Computing in Prison

The hardest I ever laughed was the day I got a letter forwarded to me from *Computers in Physics*. It seems the warden of a federal prison in Florida had refused to let an inmate receive an issue of the magazine because, the warden stated, "It contains *explicit* computer code." It kind of sounded pornograpic. He cited some C++ code in an article in my Department. Every colleague I showed that to couldn't stop laughing at the warden.

EVALUATING CLIMATE-MODELING CODES

I worked for the Program For Climate Model Diagnosis and Intercomparison (PCMDI) for three years 1998-2001. Our main product was graphical— those pictures of world-wide temperatures with lots of colored contours, etc. But there was also a data-processing component. I enjoyed it at first because it was a chance to work on a product meant for personal computers and high-capacity workstations, including using Python. No more supercomputers or classified papers.

The United States had a national model of its own, headquartered at NOAA in Boulder, Colorado. Livermore Lab also had a model, as one of about 30 codes made around the world. I worked on CDAT, a tool for analyzing the output of these climate models, which were in a format the climate community had agreed upon. The people in PCMDI were good scientists, but compared to the computational physicists they were beginners at computational science.

As I became familiar with the national effort, I was shocked. The code-development practices and testing were from the 70's. Individual packages were "glued" together at NOAA , exactly as in the Tokamak Systems Code failures.

I gave talks about Basis to this group about the good effects of building their individual

[28] P. F. Dubois, "Goodbye to All That", Computing in Science & Engineering, v. 10, #4, Jul/Aug 2008.

models in that environment, but the general feeling was that they didn't want to put in the energy to change their methods. By this time I was focusing my message as the need to use methods that enabled a code to stay correct in an atmosphere of continuous change.

Worse, the researchers might have a "bake off" for some portion of the physics, such as "cloud formation", which depends on atmospheric properties, topography, and vegetation, and then they used the best one of those without regard to how the overall (complete) model reacted. Since simulation involves making assumptions about how all the processes interact, the overall prediction might be become less trustworthy if you made the clouds "better". If you can get all your modeling to first principles, without a slew of tuning parameters, that's one thing. But unlike laser fusion, climate modeling was miles away from that.

For example: the models did not conserve energy, i .e., they lost heat overall, as the model must solve an equation with a numerical method that advances the climate over time. With realistic input data, the computational "Earth" ends up a snowball in a hundred years. Therefore, a climate model adds a lot of tuning parameters (I call them "knobs") including energy input, to make up for the imperfect numerical processes involved. By turning knobs, one can predict as much global warming as desired. And desire it they do, for that way lies fame, fortune, and funding.

Suppose you set these parameters to fit the model to some past data. Then, having found a good fit to past data, you extrapolate a hundred years ahead. I was surprised when I learned that a climate model is not solving climate equations. Rather, it is a weather model. "Climate" means "Average Weather." A climate model is just a relatively low-resolution weather prediction code, so you can run it many, many times to compute average weather using slightly different starting values each time.

Lacking the computer and data-storage resources to do that, and also lacking a measure of the variance in the initial data, they accepted the first run that comes out. And that initial data — much of it is only a guess. The temperature of the surface of the sea, for example, only goes back fifty years to some satellite data. Similarly, one would be hard put to accurately input the amount of moisture in the ground everywhere.

So how valid is it to then extrapolate for a hundred years, or worse, restrict your attention to a fraction of the globe such as the Arctic or California? The software running could make you a picture, with lots of colors and details, and you can put the pictures in a newspaper.

Science consists of making hypotheses and testing those hypotheses with experiments, even though Climate is something for which you can't do a lot of experiments. But if we make predictions based on computer models, we must not fail to observe the accuracy of those

predictions later. Unless this process is completed, we can't have any confidence about the model's future predictions. By 2017, the average temperature predictions of the combined world climate models were just wrong.

At the time I was in PCMDI, the climate-modeling community was pushing the Kyoto Treaty, which called for drastic carbon emission reductions but only in a few rich countries. No restriction for China or India, where major emissions originate. To implement the terms of this treaty, employment in the U. S. would be greatly harmed, the price of gas would increase, and outsourcing of jobs would be assured. The proposed reductions would, according to the combined world models, and the U. S. Model in particular, increase the temperature by just one-half a degree Celsius in one hundred years, they said. Advocates for Kyoto admitted that such a treaty was really just "symbolic". It was upsetting to think that some single-mother waitress would pay for this action based on untrustworthy calculations. At that time the codes would simulate one hundred years and never exhibit one "El Nino", the largest climate phenomenon on the planet. It seemed like a fraud to me.

Over the next 15-20 years, there was a change of wording from "global warming" to "climate change", reflecting the fact that "global warming" might not be happening. A great deal of effort has gone into understanding just why the models were so wrong. Computers have gotten enormously more powerful and capable of much finer spatial and time resolution, and they are able to afford to include more physical processes. I don't know what modern methods they may have adopted. I hope their models are now much more correct, but tuning the parameters to fit the data you've collected up to now is not really normal science[29]. Once again we must await the future and see how today's predictions do.

I know how they did in the past. One day I sat in the PCMDI library to try to learn more of the science, and read of an ominous future in a book from 1970: an ice-age would be coming soon.

KULL CONQUERS ALL

I was asked to return to X-Division to help with a new simulation, a three-dimensional code using Python steering over components written in Fortran and C++, the object-oriented extension of the C language. We would be developing the code on both classified and unclassified environments, which meant I could continue to do some work from home. They

[29] There is an interestingly similar argument about cosmic inflation models.

put it to me as being for the good of the country, so I made the change back to my old group, now called X-Division. The new code, called *Kull,*[30] was begun in order to do in three dimensions what LASNEX did in two, and to become the main code for Inertial Confinement Fusion. They called me the Software Architect. I found the team to be the most delightful one possible; some had learned about steering and Object-Oriented Programming in a class I'd given some years before. Physicist Doug Miller was a good project leader, and he had assembled a superstar crew and a veteran computer science team. While personnel changed over time, here is a snapshot Doug gave me circa 2003.

Physics: Doug Miller, Peter Anninos, Armand Attia, Patrick Brantley, Pete Eltgroth, Nick Gentile, Tom Kaiser, Mike Lambert, Neil Madsen, Paul Nowak, Mike Nemanic, Mike Owen, Doug Peters, Sisira Weeratunga, Mike Zika.

Computer Science: myself, Pat Miller, Martin Casado, Charlie Crabb, Susan Hazlett, Jeffrey Johnson, Kathleen McCandless, Chuzo Okuda.

With Kull, I had a chance to do something meaningful towards code correctness. C++ allowed for implementation of some assertion statements that would bring to bear part of Meyer's object-integrity protocols. If, before submitting a change to the Main, or source "repository", a physicist could assure that the new components were correct, and that some basic tests of integration were successful, the chances would be good that one physicist wouldn't "break" the Main by adding sections of code. That allowed scientists to keep their "branch" of the source code up to date, rather than getting months behind and face a daunting task of integrating their work back to the Main"

To my gratification, this group supported my ideas and based the work on my steering ideas, Object-Oriented scientific software, and correctness. Soon it was routine to run a thousand tests using parallel computers as a precondition of updating the repository. We had the best time together. We even had a bowling team. I think Kull qualifies as one of the most sophisticated machines ever built by man .

CASC Is My Matrixed Home

At one point I had become an artist in residence in the new Center for Applied Scientific Computing (CASC). I had a chance to mentor some young computer scientists and complete the circle by sharing my experience on what the Livermore "Programs" are like and what they need. CASC was led by another good leader, Steve Ashby, and finally the computing

[30] Kull The Conqueror is a fictional brother of Conan The Barbarian. All the team attended the debut of the movie. The stereotypes about the comic/fantasy/science-fiction interests of the physics and computer worlds are quite true.

part of Computational Science became a proper discipline at Livermore.

CASC's heritage derived from the Numerical Mathematics Group; NMG and Statistics had merged, and then split again with NMG joining some computer scientists to form "Computer and Mathematical Research". I was asked to be the founding leader of that organization in the Spring of 1985, until I took a leave to welcome my adopted son home from Korea. In my absence someone did some politics, and on my return learned I wasn't going to be in that job. I asked Lokke if I could escape from the political mess that the Computation Directorate had become, and in typical Lokke fashion, he said sure without any further discussion. So I belonged to Physics programs up until I went to PCMDI and discovered Computations was still a quagmire. When I started work in LASNEX, I was in fact the group leader for the Computer-Science team without being myself in Computations' Applications Programming organization. I think it was when I went to Kull that I was put in CASC.

Improving Testing and Correctness

Object-oriented programming and assertions had been about correctness. More than simply being a way to have a program correct at its start, I realized these ideas were really about helping scientific programs *stay* correct and weather the continuous change effected by groups of scientists continually adding to a major piece of software. I had noted that in one year there had been 75 major changes in LASNEX, a year we thought of as a quiet year. Without some plan to ensure correctness, a large program becomes so difficult to change that a new person finds it too difficult to understand and cannot work on it. After an addition, the program may still run but get incorrect answers. The idea of starting over is then put forward; this has happened in the past at Livermore Lab and everywhere else.

In the section "Thoughts On Software Development" I discuss correctness at length. During my work on Kull, this became my main interest. As I worked on my automatic testing system (ATS) , which I wrote in Python, I began giving a series of talks on the topic of testing and correctness that included a picture of an iceberg, suggesting the idea that these techniques would let a programming change "melt" only a small part of the iceberg, which would quickly re-freeze (stabilize) with testing.

My final trip was sort of a farewell tour, with talks in Paris at a government aerospace institute, a university campus, and then on to Charleroi, Belgium for an invited presentation at the European Python conference.

VIII. RETIREMENT COMES TO US ALL

"My goal is to retire right before senility."— Elon Musk, interview, 2012

GOING OUT ON TOP

Retired in La Mesa, CA, 2018

My big fear was becoming an old fart who had been great once but now didn't understand the new ideas and was resented by the younger people working for him. Better to go out on top. I felt with Elon Musk that, "I didn't want to be the old grandpa who doesn't know how to email."

The Division agreed that it would take me back after retirement at 40% time for another two years. This was a good deal for both sides: as a retiree, I didn't get benefits, and my cost to them was very much lower.

So in June 2005, after 29 years at the Livermore Lab, I retired. There was a very nice retirement party for me. Most of the old gang from the Numerical Mathematics Group was there. I'd hired three people there who had each had a big impact, mathematicians Milo Dorr, Mark Seager, and Jeff Painter.

When the two years were up, almost all of it spent on the Automatic Testing System, including adding more parallel-processing capability, I stopped working. In 2010, they told me the testing system was a great success and that they had a list of additional capabilities they would like me to add to it, so I did that part-time over about a year and a half. The result was a tool that runs thousands of tests at once on massively parallel hardware— tests of a program that is itself a parallel program.

* * *

I Can't Stop Computing: My Apple App

During the next few years I did almost no computing, but I took an online course from Stanford about *Swift*, Apple's new object-oriented language for iOS platforms. One of the ego-boosters was watching the Stanford nerds, some of the best in the country, asking language questions I thought were elementary. I was also able to take commercial courses[31] on *Udemy*, an online technical-education platform, not to mention online courses on my hobby of baking. It was just thirty years from having no net to all the free or cheap education you can use! And then came *YouTube*!

For a hobby, I had gotten into Duplicate Bridge in a big way. I played at two clubs, one in Livermore and one in Pleasanton. The Pleasanton club folded, and a second weekly game in Livermore was begun to replace it. I took a Bridge Director's course, and I was eventually given ownership of the Livermore club.

We had no tournament clock, and the commercial ones cost a fair amount. I decided to write an *app* to turn an old *iPad* into a full-featured bridge director's clock. I released it in the Apple Store at $10, on my 72nd birthday in 2017. It sold a couple of hundred copies over the next three years, mostly overseas. As of now, that looks like my last programming hurrah. Not that I don't dash off a little Python now and then.

[31] Including how to use the software, *Scrivener*, that I used to write this.

IX. THOUGHTS ON SOFTWARE ENGINEERING

TOOLS FOR LARGE-CODE DEVELOPMENT

Besides the program itself, a large scientific simulation lives in an environment of tools that enable a team to all work on it at the same time. Such teams can range from 2 to say 20 scientists. Making this easy for the scientist is a very hard problem.

In scientific programs the scientific content (as opposed to the user interface, database access, graphics, and so on) is in nearly self-contained "packages" written by 1-3 people who are developing algorithms for a certain kind of physics content, such as hydrodynamics or heat transport, as I described in discussing MERTH. There is an underlying set of "state" values that are kept by these processes; many of these are shared and altered by more than one package. The other values are temporary.

It is important that the scientists assigned to each package can work independently and then merge their new changes back into the main line of the program. Sometimes the new code can be done incrementally, or it might be the better part of a year before this can be done.

When I worked on MEG, I started with the public source code and worked on an assigned physics package for several months. I thought all was well until I let the test program run a long time and found that after agreeing bit-for-bit for a long time, the answer began to diverge from the standard version very slowly, and then more and more rapidly. I ran the debugging tool for a week and finally determined that one subroutine was getting the wrong answer on the 351st time it was called. A term in an equation that was initially computed as zero had stayed that way until, on the 350th call, a material had gotten hot enough for this term to be bigger than zero. The term was added to a state variable. Unfortunately one line was missing from the code: the line to set this intermediate term to zero when you entered the subroutine, so it began accumulating the previous value with the new one. Today, the compiler can warn you of such an "uninitialized variable", but not back then.

With great trepidation I went to George Kramer to report that I had found an error in MEG and so all of their results, I supposed, were wrong. George looked up from his desk and smiled and said, no, everything was OK, because "We already fixed that." Nobody had

thought to tell me, and the system did not have a way to tell me that a change had been made in the lines I was working with. If I had not noticed the wrong result, say by running only 349 calls, and had I put my source back into the main line, I would have reintroduced this error (a so-called “regression” error).

The “tool chain” provided to the developers needs to:

- Manage the sources, including merging changes with those of other developers.
- Build new versions.
- Test new versions for both correctness of changes and quality assurance for major releases and new developments.
- When a bug is found, a test must be added that would find that bug if it is reintroduced.
- Manage public release facilities, including stable and experimental releases, and communication with users about updates.
- Track bug reports and requests for new facilities, and help manage the team to be sure these are addressed.
- Assisting developers in finding things in the source code, so they know what other developers might be affected by their changes and coordinate with them.
- Provide access to mathematical, database, and graphics software.
- Provide a means of documenting the code.

When I first arrived in the LASNEX group, here is how it was done:

- A public and an experimental version were in use. Stored with these were the libraries of compiled “binaries”. To make a new version a developer would edit, add, or delete files of source code, compile it and load the program into a new executable using the new binaries followed by the old ones, so that new versions were used in preference to their predecessors.
- The developer would run his own tests of the physics he was changing, and when merging back to the Main line, would run one or two integrated tests to be sure this new physics hadn’t accidentally broken something else. But the developer had no access to run these “private” tests on other people’s packages to be sure they still worked. More social friction would ensue when they didn’t.
- Changes between the original version were stored along with the public libraries. As time went on a file might accumulate a chain of changes.

- When building a particular source file, the source corresponding to the last public release was modified using the stored differences between the standard and the current version of the file.
- The compiling and loading for the new version took place on the supercomputers. Since these were an overused resource, it sometimes took three days if you needed to change the entire Main line. The entire process was serial, one file at a time.
- I was the bug-tracking and reporting system for computer-science components, and the individual physicists kept track of their areas of responsibility.
- There were no tools for searching the source other than the text editor.
- There was a program-wide database library, and the computer center provided a graphics library; the Numerical Mathematics Group provided mathematics routines, or authors would type in algorithms from books, sometimes using inferior algorithms.

Unhelpful behavior was encouraged by the great time-and-effort overhead on modification. Modern practices such as *refactoring* code did not happen. If adding a new variable to a common block was necessary, developers put it on the end of the list, even if logically it was related to existing variables in a different part of the list. I viewed solving these problems as crucial, since otherwise the code would accumulate debris akin to hardening of the arteries. Eventually, a revolt takes place as some new scientist decides this code is just too hard to work on and starts another one.

The “open source” movement had produced some tools for source management. In most cases I am a big fan of open-source software; but in this area the open tools were far behind commercial ones, in particular a product called *Perforce* from an Alameda start-up. Naturally, getting a substantial outside purchase done was a headache, and Division management naturally wanted to avoid a different system being used in different codes, thus requiring retraining when a physicist moved to a different project. Perforce was relatively inexpensive, and the company still flourishes today. From the first, the company supported free individual home use and open-source projects.

A tool like Perforce stores the latest version of each file and remembers the changes in case there is a need to go back to the older ones. Therefore all the overhead to reconstruct the file to be compiled is eliminated. We had workstations now, and they were a very much better place to be manipulating text. We were soon building the Basis version of LASNEX for both workstations and the supercomputers, doing everything on the workstation except the actual compile of one big file of standard Fortran per package on the supercomputer. The time to

"build" had dropped from three days to three hours and got even better as we started to do parallel compiles.

However, the Resistance has learned it isn't futile to resist a change that you don't want to be imposed-upon from on high. The spread of Perforce outside the Weapons program codes was stymied by some unethical behavior in the Applications Programmer Division management.

As the open-source Python Language grew, as described in the next chapter, a better tool for tracking bugs and requests for enhancements to Python became necessary. Python held a contest, and the winning design was implemented. I spent some days at Google with other Silicon-Valley volunteers helping improve it, and we installed it for our use at Livermore Lab. It is a great example of a successful open-source project. Since it is free, people can learn it in college and grad-school projects.

I also wrote tools for searching the source text of LASNEX that increased developer's ability to understand and locate parts of the code. Perforce could be scripted so that if desired, searches could be confined to sets of sources that the developers understood.

Finally, tools to coordinate the testing and submission of changes to the source greatly improved quality. As the tools began to take advantage of parallel processing, by the time I retired we could run thousands of tests in a half an hour. Social stresses caused by one person interfering with another's work greatly decreased.

What Are The Barriers To Tool Adoption?

Computer scientists constantly want to add development environments or tools for software engineering. Federal funding agencies are all too eager to fund them. Most such tools fail to achieve widespread adoption. Barriers to adoption include:

- Availability: concerns the tool may not be available on the desired environment, present or future.
- Longevity: concerns about whether the tool will continue to be available, and available on new platforms when needed. Will the tool creator continue to maintain it and improve[32] it?
- Control: concerns about a lack of control if the tool is used. One cannot dismiss this as flaw in the code author's character. The concern has a strong rational basis in that the author must depend on the tool's authors for service— be it to fix bugs quickly or agree to add needed features or make it available on new platforms.

[32] A potential customer asked me if I expected to spend the rest of my life doing Basis. I replied that I probably did not, but I did have a succession plan, and in the end that plan worked.

- Payoff: concerns that the tool may not return gains in productivity or correctness sufficient to offset the time it takes to use it and, importantly, the time it takes for training team members.
- Interface: concerns that the tool will be too "computer-sciency", i.e., needlessly complicated for the people who must use it.

I believe that funding approval should depend on a proposal demonstrating an awareness of these factors and a plan to deal with them.

Keys To Program Correctness

The term **correctness** has two meanings when applied to scientific software.

The first meaning of **correctness** is the plain one, get the right answer. Unfortunately, it is impossible for a computation to "get the right answer" exactly when the numbers have to be computed using limited precision. For example, in a computer, 3.0 times (1.0/3.0) isn't exactly 1.0. After millions of such "floating-point" operations, those little fractions can add up. Geometric calculations with small angles are particularly prone to quite large errors. In the early 1980s, a professor at U. C. Berkeley made a name for himself showing people that navigating an airplane with the exciting HP calculator with "only" 12 digits of precision could get you wildly off course— unless you looked out the window, critics pointed out.

Designers of numerical algorithms have to design procedures that do not have problems with "precision" in this sense. Sometimes this can involve very difficult theory; my first work for Livermore Lab was in fact uncovering an error of this sort in a parallel algorithm for a linear-algebra problem. A "numerical analyst" is a person who does this kind of work. Assuring correctness in the program obviously involves a robust set of tests and careful numerical analysis.

The second meaning of **correctness** is be sure the algorithm is coded correctly. For example, ensuring that you don't have a minus sign in a formula instead of a plus sign, or an error in the logic, or a variable name mispelled, or a mismatch in the caller / called sequence of arguments in a library routine.

Keeping Programs Correct Is Difficult

It is a separate, and much more difficult matter, to **keep** the program correct in the face of changes to the sources. Scientific programs change constantly; if they don't, they die. Having written a function and having assured it is working correctly in a test routine does not insure that it will remain correct. We work with hundreds of changes such as:

- A new call to the function may be added elsewhere with the arguments not passed in correctly, such as listing two of them in the wrong order.
- A call by a section authored by a different person may have an argument that is out of the range of validity for which the function was designed. Perhaps an approximation works for hot water but not steam, for example.
- The new coding may be fast but not accurate.

If the developer is lucky, the program halts when the bad call is made. In the nightmare case, it returns an answer that is modestly wrong. As I mentioned in the discussion about MEG, I worked on a a routine that didn't "go bad" until the 351st time-step, and even then started to be wrong at first in the 12th decimal place. It seemed to work upon limited testing, but as the test proceeded, the error got worse.

Bertrand Meyer devoted his research to this question, and came up with a very practical solution, which he called "Design by Contract.[33]" While full implementation of this concept was not practical in any other language but Meyer's Eiffel, doing what one could in other languages using preprocessors was in practice a great leap forward. When the program goes into production, these checks are when building the program after final testing and hence incur no extra run-time for the end user.

Meyer pointed out that the big advantage of reusable libraries designed with object technology was in increasing correctness. Above and beyond not writing that code from scratch, there is the following benefit: if that reused code is already correct, and you checked your calls to it by means of assertions, the ensemble was very likely to be correct.

I taught beginning programming at a one-day summer workshop at Mills College in Oakland in about 2016. The College was recruiting women who wanted to learn to code. Apple had produced an entire learning environment for its Swift language. I had the women use this environment, and they were able to immediately produce fancy but correct programs using the Apple-supplied component libraries. I was pleased to see that my predictions of this future way of programming had come true.

While I'm sure the state of the art will keep moving, we have come a long way since I began simulating dandelions in 1971.

* * *

[33] Meyer, B., *Object-Oriented Software Construction*, Prentice-Hall, 1994; 2nd Edition, 2000.

How to Succeed In Tool-Building

It is not merely the technical virtue of a tool that determines its success. Successful tool-adoption depends on a social process. Hundreds of "computer-scientist" ideas are not used because they do not ensure that individual developers have a better day at the office.

For example: Testing regimes tend to be imposed by management in order to have confidence in any changes to the program. One of Livermore's large codes had a test problem that was equivalent to standing a pencil on its tip and watching it fall— if you changed the calculation then the pencil fell very differently. Unfortunately, any meaningful change to a program, even if correctly implemented, changes the answer. Tests like that make no sense.

On many projects, the lack of any sort of automation for testing means testing simply does not happen. Real testing must be in depth at the unit, the package, and global levels. Testing regimes then particularly need to be automated and easy to use, and the tests themselves must be easy to create. The role of team management is simply to require this testing be satisfied before changes are committed to the Main line, or repository. That extra step can be made acceptable to the authors if the testing is fast and they realize this regime will insure that they will not get "bitten" by a colleague's faulty change.

Having such accidents where faulty changes by one person create mystery failures for others are not merely a waste of developer time. Such incidents are debilitating to team morale. People have an instinct, it seems, to impute malice to others when they have simply made an error. I offered this thought to my teams when this happened: "Don't mistake incompetence for evil intention." I should make a motivational poster out of it.

Therefore, in designing my testing regimes, the goal was to enable different levels of testing, to make tests easy to add or modify, to have executing and reporting on the tests be automated and to make failures announce themselves clearly. Most importantly, the "private" tests that developers use on their pieces must be included in testing by other members of the team who left to themselves would not know how to execute such a test nor interpret its results. Once I did that, there was eager adoption and compliance with the regime— it became socially unacceptable to commit source code without doing the required tests.

Caste Dynamics Need Management

Computational scientists are also professional programmers. "No, I'm not!"each might say. "I'm a scientist." But this is a thought from the past. As I point out in my most-cited

article,[34] the act of programming is now an integral part of their professional lives. They now program and are paid while they do it.

Social status is at work here. The scientist thinks he is more capable than computer scientists or any "programmers". However, a need to keep himself in a superior status may wreak havoc on the group dynamic necessary to the care and feeding of the Laboratory's large computer programs.

Teams of people contain computational scientists, programmers, computer scientists and pure theorists or even experimentalists. All of these callings and their supporters contribute to the success in these large efforts. This may not be easy for the scientist. Each has been the smartest person in the room most of their lives, so to have 20 of these people on one project may be taxing in social relations. But a *team* is the only entity that can produce programs of the size and accuracy required.

Even on a one- or two-person project, scientists will benefit from devoting some time to keep up with the latest tools and ideas. It's a new branch of science: they must nourish their "inner programmer".

[34] P. F. Dubois, "Ten Good Practices in Scientific Programming", *Computers in Science and Engineering*, v1. #1, Jan/Feb. 1999.

X. WORK ANECDOTES AND SHORT THOUGHTS

MY LIFE IN MIDDLE MANAGEMENT

Early in my career I became a "Group Leader", which at Livermore meant something like a Squad in an army, 8 to 14 people. All my management was as at the technical level as Group Leader, or a Project Leader, or leader for the computer-science part of a physics project.

Managing a group of mathematicians is maddening. They are idealists, as I should know. Once I watched my brother Philip, Chancellor at University of North Carolina, Charlotte, try to explain to his Mathematics Department why they were not getting the resources they wanted from their legislature. They somehow thought this involved *facts* and *logic*. At the Livermore Lab, the resources had to come from Programs, and hence it was necessary to be of some use to the biggest programs. But to do that, you had to get out there and be involved, and that just was not a good fit for introverts and those who liked to work alone. The best we could do was to get those people to work alongside the few members who did go get the work.

The Buck Stops Here

My job on LASNEX involved considerable management since the team was about 10 application programmers and a technician.The technician, whose name was Al, helped our customers get things done with the central Computer Center. His performance, which had been good, began to deteriorate. Finally it got to the point where I felt the need to do something. At the Lab, leaders don't usually fire people, they ask them to find another assignment. I did that, but shortly I learned he'd broken some bones in a seizure, leading to the discovery that he had a fatal brain cancer. I kicked myself for not having had him get medically evaluated earlier.

In another management situation, I had someone on the team who was the nicest guy in the world but not doing that well. I saw a blurb about an Asian Leadership conference in Los Angeles. The theory was to help Asian professionals express themselves more forcefully in work situations. Whether there was anything to that or not, I thought at least it might motivate him because he might feel I saw potential in him. But after the trip he became

depressed, and his performance got worse. He told me that all the other people in the class had seemed so bright and capable and were younger than him, and he felt like a failure.

On every assignment I had found him coming to me for help, which was odd— usually the boss is the last person you'd want to bother. The truth was, the nature of our work was so very difficult that even this well-intentioned, well-educated, intelligent man could not do it to the level we needed.

When I finally asked him to find another assignment, he found a good job outside the Lab. To my surprise, everyone in my group came around to ask me what took me so long, explaining that he'd been driving them crazy asking for help. No one had ever complained.

I also noticed that some people were just the greatest with two projects but not with three. And I was good at not annoying a group with too much management-oriented fluff— since I hated that. I felt it was my job to keep the extras from above off their backs. I only held meetings I needed and trusted they were making progress, so I didn't ask for frequent progress reports. Being involved with the technical details of everyone's work is the essence of technical management.

Leadership, Technical or Managerial?

The Lab management is always trying to sell the line about the management and technical-leadership tracks being parallel. My wife tells me this parallel track was also promised to her Technical-Writer colleagues. Of course the tracks are not parallel— one of my computer science peers who went into management always earned more than I after that, even though his accomplishments, never impressive or programmatically relevant before he became a manager, were virtually unknown afterwards.

In 2019 Livermore Lab awarded physicist and long-time employee George Zimmerman a "Distinguished Member of Technical Staff", designation, a new award which they said "was created to serve as a career ladder for Livermore scientists and engineers within the Science & Engineering classification structure. It appropriately recognizes excellence by distinction and compensation while allowing the honored recipients to continue to deliver science and engineering solutions to critical mission areas of the Laboratory."

Better late than never. But I have my doubts about these promises. George should have been the highest paid person at the Laboratory already.

Managing Geniuses Is Easy

Jim Crotinger and Brian Yang have doctorates in Physics from M. I. T. Jim had worked with me in M-Division and ended up helping with Basis LASNEX. He ended up working in Data Science, which has of course turned out to be the next big thing.

Brian was a laser-fusion postdoc having difficulty seeing a future for himself. There simply were not enough physics jobs, so having multiple postdoctoral positions was common. Some physicists were bailing out and going to work on Wall Street, there being considerable overlap between physics simulations and financial simulations. Indeed, a short time later physicist David Kershaw left to go to a major financial house in New York. Brian worked with me into becoming a more computational-oriented physicist.

As a management strategy, I advise hiring only physics doctorates from MIT in all positions. Oh, George Zimmerman is from Cal Tech, so that's another good choice.

Michael May's Hobby

About a year after arriving at Livermore Lab, I was crossing the street inside the classified portion of the Lab when I saw one of the luminaries of computing, George Michael. George stopped me and said, "You did algebra, right? You know Galois Theory?"

I confessed I did, although it had been seven years ago, but I surely knew more than anyone else around.

"Well, I want you to call up Michael May and make an appointment to see him. He wants to know about Galois Theory. I don't know why. Maybe it is for cryptography, but I'm just speculating."

I had to ask who Michael May was. Michael May was the Director Emeritus, originally from France. Being asked to go see him was right up there with being asked to go see Edward Teller, George explained. So I made the appointment— Dr. May wanted me to come by at lunch time. Meantime I read up on practical uses of Galois Theory in cryptography.

Came the day, I met with Dr. May in his office. He had another man with him. May said, "We have been reading Galois on our lunch hours together and have some questions."

Reading Galois?? In the original French?? The paper Galois wrote all night before being killed in a duel by an *agent provocateur*?? The one nobody could understand for 50 years?? The same. They said they were just reading it out of intellectual curiosity.

Some people are just smarter than you are.

* * *

Security Clearance Tales

My original security clearance had been granted in 1975 and renewed every five years thereafter. Back then the **FBI** checked you out. An FBI agent visited the house next door to ours in Las Vegas, N. M., where he interviewed our scruffy, long-haired neighbor, who as it happened was the local drug dealer. The hippie came over afterwards to our house, quite excited. “Hey man, I told him you were cool, that I never sold you nothin’,” he reported to me.

I could potentially lose my job or worse for a myriad of causes related to security rules. I called my security badge “the instrument of obedience”, after a Star Trek episode where the aliens put something around your neck to give you a punishment when needed.

Security clearances did not take long to process until Patty Hearst was kidnapped in California. To solve that crime, the FBI pulled all the agents off clearances and started training people in the Office of Personnel Management to do the clearances. But meantime, they got way behind. It took three months for my clearance, but a year and a half for my wife’s just a few years later.

Livermore’s Employee Benefits

One of my earliest Lab recollections was someone explaining to me that if I ever needed equipment that I should ask for it. “Your time costs a fortune compared to anything you need.” Despite that encouragement, I never tried ordering a gold bar from the Supplies Catalog.

Vacation started at 3 weeks a year and increased to 5 weeks a year, with additional sick leave. Some people took a sabbatical leave after seven years, to visit at some university.

The Lab had a medical department, its own fire department, and its own police. It had its own lake, and its own swimming pool (listed on the map as a “fire reserve”). There was a bunker for some radioactive material. Out by one gate was parked an antiquated armored vehicle. We called it “the tank,” but it was not one, just a small Armored Personnel Carrier. In reality, it was an early-warning device. The rumor was that any terrorist attack team would doubtless blow it up on the way in.

When traveling out of San Francisco airport, employees used to be able to take a light plane from Livermore Airport. Once I was the only passenger and got to sit next to the pilot and listen to the air traffic control. We landed on the regular runways and taxied off to a building

called Butler Aviation, where they would give me a ride to the terminal. The only downside was fog: if it didn't lift in time, you had to drive like mad across a bridge to make your flight.

Livermore Lab had no child-care services when I began work there. I served on a committee that helped start a child care facility at a nearby former elementary school. Employees could send their children for a fee, and my own children went there briefly. The operation was subject to State rules for such facilities, such as ratio of child to care-giver. It was shocking to learn how little the pay is for child-care employees; we did our best to at least pay the top end of the range.

Bertrand Meyer Was My Mentor

Professor Bertrand Meyer, now Emeritis at ETH is one of the architects of Object-Oriented Programming and modern software engineering. Although I mentioned some of this already, there is much more to tell. I got many ideas from him, and we became friends.

Meyer's contributions to computer science began with his interest in the question of how to write programs that were provably correct. He thought about how the programmer could write a specification that an implementation was designed to fulfill and test that it was doing so. This led to his design of the *Eiffel* language and its facilities for "assertions", statements meant to be the programming specification, and "invariants", statements about an object's integrity. When testing found no more bugs, the program could be run with the testing off, to achieve maximal speed. But when a bug is found, its fix should involve the addition of one or more assertions that would have prevented that error. This method does not achieve formal provability of programs, but in a practical sense it comes very close.

Eiffel is a pure object-oriented language, the first major OOP language that did not derive from a procedural language, and hence Meyer did not have to compromise its ideas to maintain compatibility with archival sources. Meyer's ideas and facilities in Eiffel for assertions became far more important than Eiffel itself.

As my interest in OOP began, I was lucky enough to attend the first course Meyer offered in Santa Barbara. His book, Object-Oriented Software Construction,[35] was a revelation. He had many ideas about concurrency and testing that influenced me as well.

[35] Meyer, B., *Object-Oriented Software Construction*, Prentice-Hall, 1994; 2nd Edition, 2000.

EiffelMath

Although my interest in Eiffel was limited by having to use tools available on our supercomputers, I did attend *TOOLS Europe* and *TOOLS USA*, conferences Meyer ran focused on Eiffel and ideas about correctness and OOP.

Livermore Lab allowed me to do outside work on my own time. I did have five weeks of vacation a year to work with besides nights and weekends. Meyer asked me if I would be interested in writing an Eiffel interface to a major mathematical library, NAG, written in Fortran. The customer he had for this library was an investment firm in Paris. Meyer told me he had found a Polish refugee in Eastern Canada, Jan Sabinski, to do the implementation, but wanted me to figure out the right way to design the interface. He gave me a copy of Eiffel for my home computer for the purpose of exploring designs.

Since real money would be on the line, the investment firm had a fiduciary duty to create a code that was correct. The NAG mathematical library was a well-established and trusted component, so I needed to design an interface to it that was in the Eiffel spirit and used the assertions of correctness in the language. I invented an approach to constructing mathematical software in an object-oriented way. In this case, an object might be a "thing" that could solve a differential equation. When used in a Fortran code, such math library routines were very hard to use; mine were easy and reusable. This work cleared up for me the question of why mathematical software had been so hard to share in big applications.

When the library was finished, I wrote it up and it was published by Prentice-Hall in 1997[36]. In 2021 you could buy a used copy for $16.97, but I think they sold it for about $40 originally. I earned $1500 for the book, which considering the hours it took my wife and I to prepare camera-ready pages, it would have been time better spent working at McDonald's.

To report to the customer that was a bank, in Paris I stayed in the five-star hotel "Georges V", just off the Champs Elysée. We walked the two blocks to the investment house. There was a big room in which individual economists were learning to program in Eiffel. In order to keep them on the job, lunch was brought in every day from nearby restaurants. French food: it was great stuff. People would serve themselves and go back to eat at their desk.

The library did enable correctness; I recall there being exactly two bug reports, one of which turned out to be in NAG.

[36] P. F. Dubois, *Object Technology for Scientific Computing*, Prentice-Hall, New Jersey, 1997. 288 pp.

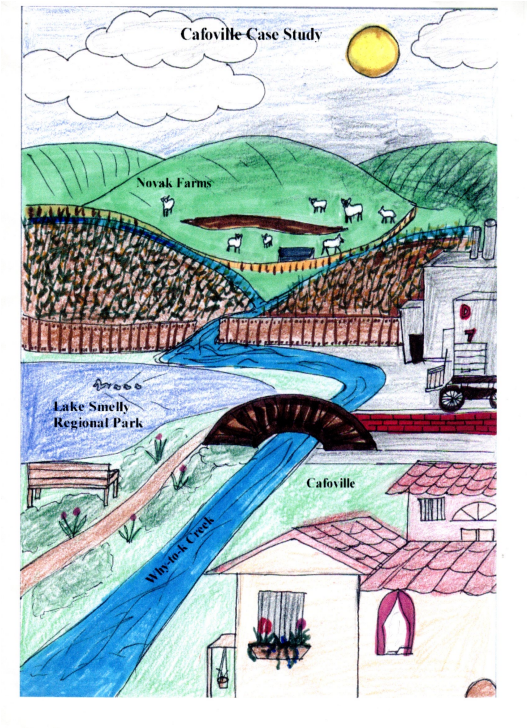
Talk at the Eiffel Tower

Yes, there is a Conference Room inside the Eiffel Tower. I liked to attend the International Eiffel User's Group Conferences. At the fifth in 1990, "The Evolution of Large-scale Scientific Computations", I gave the Keynote Address. I spoke a few words in French, saying that it was to honor my Belgian grandparents. Afterwards I was introduced to an enthusiastic young researcher from Belgium who had appreciated my doing that, if not my terrible accent. The conference was typical of Meyer's flair for Eiffel-event locations in Paris. Over the years there were conference dinners at a Moulin-Rouge type theater on the Left Bank that had turned out to have been designed by Eiffel himself; at the former carriage house at Versailles; at Musée de l'Homme at Place du Trocadéro across the Seine from the Eiffel Tower; and a restaurant in Versailles.

Teaching OOP to the EPA in D.C.

Meyer received a request to develop a course to cover object-oriented design for a branch of the Environmental Protection Agency. The course was to be several days long and be delivered at three different EPA facilities. The point of the course was to help them design a code to simulate both the operations of mass hog production and the safety of the handling of biological waste. Meyer said he could give the course on one occasion, and he had someone else who could do another, but he knew nothing of scientific simulations and asked if I could help. I agreed to design and deliver the first course in Washington, D. C.

The subject matter was interesting. EPA wanted the course to explore the specific topic of over-flowing urine-retention ponds. Pond overflow, I learned, could cause environmental damage, particularly if large rainstorms occurred. Normally some of the waste could be used for attached farming areas, but the salts in the effluent limited that, and if you were caught with a full pond in a downpour you could violate the law. "What happens when you do that?" I asked, and learned that it depended on your type of permit. What actually happened to the environment, they didn't care about one whit— just whether they could fine somebody.



EPA case study cover by Claire Dubois. The creek is labeled "Why-to-k" Creek.

My daughter drew the cover for the case-study book, and off I went to Washington. Due to flight delays, I arrived at the hotel two blocks from the EPA about 8 P. M., famished. I asked the desk clerk about getting something to eat, but he said there was no restaurant in the hotel. I looked longingly out the window at the McDonalds across the street. The clerk said, "You don't want to go out there now. Too dangerous."

The next morning I walked the two blocks to the EPA and found I was crossing the front of a housing project. My host greeted me in the parking lot. The first thing he said was, pointing to a nearby parking spot, "Someone got shot right there last summer."

I Promote Open-Source Software

I am a big supporter of *open-source* software. The main reason is selfish. It turns out that if you make your source available to the public, in return the public can find errors and write improvements for you. **The result is software that is more capable and error-free than proprietary software.**

For example, Microsoft Windows is proprietary, and Linux/Unix are open-source. Linus/Unix, and systems like MacOs that are based on them, are much less exploited by hackers and have many fewer bugs. I was determined to make Basis open-source as well.

Besides the participation of the public, making Basis open-sourced defused a frequent worry of potential users. If I or the Livermore Lab lost interest in supporting the system, I could assure the users that they were free to modify the software themselves. The only thing they could not do was sell it.

It took a couple of years of effort to convince Lab management of these benefits. We had a Technology-Transfer division then, and their interest was in selling things so they could justify their own budget. It took a lot of persuading to convince them that nobody would pay money for such a thing as Basis. For one thing, most of the natural customers were at national labs, and they were entitled to get it free.

One success I had was getting the Lab to help fund a non-profit Python Foundation to hold the intellectual property as perpetually open-source. I was also allowed to contribute to such open-source projects as *Numerical Python*, and the bug-reporting software used for the Python website.

Espresso As A Security Measure

I spent a week on Lab business helping people at our sister Lab at St. Georges, southeast of Paris, use Basis. The lunch in their cafeteria was great, two hours long, and we even bought a bottle of wine to share one day. The wine is available right at the check-out, even though the cafeteria was on site.

As with our Lab, they had both unclassified and classified areas, and the cafeteria straddled the dividing line. The food service area was in a strip down the middle. In order to reduce the chance of unclassified people overhearing classified conversations, there was a double row of noisy espresso machines separating the two areas. Really, maybe forty espresso machines.

Talking With Seymour Cray

The man who designed the CDC supercomputers was named Seymour Cray. My understanding is that he left CDC because he didn't like what they were doing with the ill-fated STAR-100. He founded his own company, Cray Research, and created the Cray-1. We all fell in love with it.

The machine was a tall cylinder, with a hollow interior, and it did not quite go all the way around to form a full circle. Surrounding the cylinder part was a row of comfy padded seats that concealed power supplies and the like. Rumor had it that Cray had only one small Asian woman engineer who could squeeze into the cylinder to work on the wiring.

I had an interesting talk with Cray one day over lunch at a meeting. He said that when he sold the first Cray-1, he broke even financially. Then he sold a lot more, a whole lot more. Each machine had a serial number, and there were great holes in the serial number sequence between us and the other national labs, so we speculated there was a whole lot of code-breaking going on somewhere.

Cray said that he really considered himself a cooling engineer. The problem with a super-computer is that it generates a lot of heat, and getting the heat out of it was the main design problem, he said. Sure enough, when the Cray-2 appeared, it was liquid-cooled with the chips immersed in an electrically-inert fluid. You could see bubbles forming and rising to the surface through the transparent housing. Livermore's Cray-2 was nicknamed "Bubbles". It was the first commercial multi-processor; it had four processors, running at 1.9 GHz.

Your Equipment May Vary

In 1976 I had a *teletype* as a terminal, and within a couple of years that was replaced by a terminal that used a thermal process to print on special paper.

The *thermal paper* for the terminals came on rolls, and the used paper would scroll off the back of the terminal and fall towards the floor. This resulted in huge amounts of paper piled up, usually near the electrical outlet. Surprisingly, there were no fires. There was not enough safety consciousness until after the big earthquake, when the conditions of the offices made it hard to exit. My friend Physicist Bruce Langdon finally got ordered to clean his office when the drift of loose papers under a round table in his office reached about two feet deep.

There was a programmer at the National Magnetic Fusion Computer Center who was a rather a social jerk. As a trick, some co-workers hacked his *paper terminal* one day, so that when he came back from lunch and touched a key, it printed out a giant poster that contained an expletive, using lined-up characters to form the letters. It took maybe five minutes to print it. He mistakenly touched another key in order to try to fix it, and it printed out another poster.

The *TMDS monitors* were hung from the ceiling by brackets. Finally we all got "*glass terminals"* which solved the paper problem but left you with no written record of what was printed out, unless you scrolled back. In this case, you could print files at the computer center and go pick up the output on paper that had sprocket holes in it and perforations that allowed the paper to fold. Local printers were later installed for smaller jobs.

When we gave presentations, we first used *printed transparencies* that lay flat on a light projector. When projectors connected to the computers were added, there were many failures

connecting to them properly. I once gave an important talk on a chalk board when such a system failed.

Some members of Magnetic Fusion who visited the Soviet Union were surprised to discover their Russian colleagues carrying around *scrolls* of printout paper, because the central planners had not realized the paper would need to fold. Likewise, their IBM cards carried no readable legend on the top. The Soviet scientists could read their cards by holding them up to the light.

Parallel Computing and Its Problems

In the early 1980s, physicists Tim Axelrod, Pete Eltgroth, and I were assigned to evaluate a parallel computer the Lab was building for the Navy. We got five brand-new Sun workstations, the first of their kind, lined up in a row in one of the old barracks buildings. We wrote a simulation of how the parallel computer would perform once it was built, and the answer was, not very well, and so that part of the project got cancelled. Parallel computers were coming, but not yet. The "Sun 100"s were great, but blew their power supplies all the time.

I will leave it to the people who led the charge on parallel hardware to tell the story of its eventual success, but even so, that early work pointed to problems that have never gone away. When the physics is local, or even breaks into a lot of similar tasks, we called it "embarassingly parallel". But some physics is not local on the scale of our usual time-steps. If a process emits electromagnetic radiation, for example, it travels so much faster than the materials can move that it might as well be instantaneous. That means your calculation has to use some information from "far away", that is, data owned by other processors.

I frequently argued that some money should be spent on solving such problems and the problems of the programming environment for such beasts, but that didn't win us any prizes for the fastest computer or move your Lab up in comparative speed rankings. I coined a phrase back in the day when the thirst was for faster individual processors, to explain this single-mindedness: "Physicists are the ones who love hardware because they are the only ones who understand the word *megahertz*."

* * *

Bean-Counted Research is Smelly[37]

By the 1990s, the Lab had gone from a model where scientists could spend perhaps 20% of our time on independent research, to one where you needed to compete for money to pursue new ideas. All your activities were bean-counted.

Scientists had to spend time competing for grants to be able to explore for the future, and a great deal was lost. Bean-counting was easier on managers, who didn't have to evaluate that part of your time that you used for independent research, and it gave them more power. A manager avoided those tough conversations with those people who weren't doing promising research. Worst of all, bean-counting brought politics into it, and not the desired meritocracy.

I remember giving a talk once for research money, and a big shot said that my idea was all bullshit (*sic*), that his engineers could do it in a short time, and that it wasn't useful anyway. He was a protegé of Edward Teller, who constantly name-dropped. Physicist Tim Axelrod courageously spoke up and said that he was wrong. But in the end, the big-shot got the money.

I Tell the Future

Before that change in policy, I'd been able to freely exercise my superpower, which was foretelling the technological future. Physicist George Zimmerman had talked to me about certain matrices that arose in a calculation. They were "symmetrical", that is, each entry in the matrix A above the main diagonal was the same as its mirror position below the main diagonal. Taking advantage of that symmetry allowed solving the equation Ax = b in an efficient way.

I understood the reason the equations led to a matrix that was symmetric. I decided it might not be symmetric in the future, and as part of my "free time" I began exploring methods for asymmetric problems. I found an algorithm in the literature that had been discounted because there was a theoretical chance it would fail to converge. I decided to code it up and find out how probable the failure would be. It seemed to me the failure was very unlikely. If it did fail, we could simply drop the time-step to recover, I thought. My research seemed to bear that out.

Sure enough, about two years later, George leaned in my doorway and told me that the matrix was not going to be symmetric any more if he added certain terms, and could I start researching the problem? The new method was placed in in the program forthwith. At my

[37] P. F. Dubois, "Bean-Counted Research is Smelly", Computing in Science & Engineering, v. 7, #6, Nov/Dec 2005

retirement, George wrote on his card for me that my being ready "years ahead of time" was very important to him.

Bean-counting was a recipe for stopping creativity by people like me, who, I like to say, thought outside the box. My concern and personal charge continued to be to predict the future— what would computers be like, what software approaches would turn out to be the best, etc. I'd say I can't predict the future if I'm a slave to the present.[38]

Nora Smiriga, European Manager

Around 1980, management had a reorganization, merging two "groups," Numerical Mathematics and Statistics, into a "Section". Statistician Nora Smiriga was named head of this "Mathematics and Statistics Section".

Nora was originally from Belgium, and her English was studded with little evidences of that, chief of which was her habit of beginning a sentence with "What happens is this", which served the purpose of giving her time to formulate her sentences before transmission. She also frequently said, "What I am doing?"

A couple of times she told me something that wasn't true, feeling that there was no point telling me the truth if it would cause me to get mad and try to do something about the issue. In her mind, she was protecting me from my idealistic self, which she mentioned in one of my reviews. One time I called her on a particularly blatant lie and she said, sheepishly, "Well, you know, in Europe there is a different attitude towards things like that."

Nora and I shared a love of cats, and while the Section was working in one of the old barracks buildings, Nora and I shared an effort to feed the cats that lived under the building. Then one day we got a directive from the Lab management saying that employees were forbidden to feed the cats. Nora told me that I should stop feeding the cats and that she would take care of it alone. "They might do something to you," she said, "but they can't do anything to me."

Some years later, I learned we need not have worried about the cats. Indeed, the lab also had some foxes and now and then a mountain lion, since part of the site was bordered by uninhabited land. I stayed late one night and came out to see rabbits all over the lawns, as far as the eye could see.

* * *

[38] Nora Smiriga advised me to leave accomplishments out of my annual report if I already had enough, so that I could report them in the future while I chased an idea.

At The Test Site

My only work out in the "field" occurred as part of *Ionosonde*. A radar was to be used on one of Livermore Lab's underground tests to record data for our project. The man who ran the radar had an assistant who became ill just a few days before the test. He needed this assistant only for physical help— handing him screwdrivers and the like— if there was a last-minute problem.

I volunteered to go to the Nevada Test Site with him. We flew on the daily airplane shuttle from Livermore to the landing area. Then we rode a shuttle bus to the Site. While there, we stayed in some rooms they provided. We were issued a truck with a CB radio.

We arrived the day before the test because we had to report for the briefing in the middle of the night for the test just after dawn. The briefing took a couple of hours and was fascinating. Confirmation that the men underground in mines in the nearby states had been warned; an estimate of the cost to the government to pay damages to ranchers if the test leaked radiation on their livestock. After about two hours of such items we get to the last item on the list: the weather. The guy gets up and says "No chance". "Too much wind". Apparently having the test leak on a few cows they could live with, but not if it blew all over Las Vegas.

Nobody cried out "Why the hell did we have the briefings for two hours when you knew from the start we weren't going?" so I didn't, but I thought it. The test was rescheduled for the next morning.

The Site had a bowling alley, a bar, and a restaurant which served a legendary prime rib at a subsidized low price. The only other thing to do was to explore the desert. And there was plenty to explore. I took the truck and found my primary target, the exploded motel we had seen on TV in a film of the surface explosion of an atomic bomb, with a fake motel full of crash dummies to show the effects of the blast on civilian structures. (If you ever see the hydrogen bomb tests from the Pacific, you'll see the silhouettes of derelict World War II ships placed between the test and the camera, to study what happens to them). As a child I'd been entranced by the shock wave blowing through the motel that they showed on TV, in between practicing hiding under our school desks for "air-raid drills".

The motel was still sitting there. There was some warning sign about low levels of residual radiation in the area, but it was OK to look at the motel from the road. One nice thing about being the government is never having to clean up your exploded motels. I also found the abandoned test stand for a nuclear-powered rocket engine. Apparently it finally occurred to someone that you could not exactly blast off spewing radiation across the country.

I had yet another day of this due to another weather delay with the entire briefing repeated. Finally on the third day, the test was on.

We drove to the site of the radar set, on a hill about 10 miles from the underground test location. We had a clear view of the place. For about two hours, the radar was all warmed up and performing perfectly. Then, with less than ten minutes to go, it stopped. Our man with the screwdriver was upside down working on it like a mechanic under a car. With two minutes to go, he got it fixed.

I hurried outside the little hut and watched as the device detonated. I could see a large loud-speaker-like disk of desert heave upward and fall back in a cloud of dust, and the earthquake-like waves arrived thereafter. My sound waves were on their way to the Ionosphere.

My father, as a reporter for the Oakland Tribune, had been sent to a trench to report on one of the above-ground tests in the early 1950s. When he came home, he wouldn't describe it to me. He said it was just too horrible. I never forgot that. Experiencing the earth shaking so hard from ten miles away was enough for me to think the same thing. Whenever you hear people casually saying we should nuke so-and-so, you're hearing the cost of not testing where people can see and feel it.

Aging In Computational Science

My first group-leader position was in the Mathematics and Statistics Section.

Bob Pexton was approaching 30 years at Livermore Lab, having arrived in the earliest days. He had fascinating stories of his starting as a "programmer". A programmer then wasn't programming computers but was actually being a human computer. Bob described using big sheets of paper with grids ruled on them, taking time-steps by averaging neighboring quantities and erasing the old value to enter the new one. When the paper got too dirty, he copied everything to a new one. They even invented "adaptive" methods by skipping the updates on the "quiet" areas of the grid for several time-steps.

Later, Bob was a national expert on the computation of "special functions". Many functions that arise in nature are not simple algebraic functions but rather have to be approximated using complicated algorithms. Bob kept up a library of these functions.

When the vectorization of computation began, this presented a challenge, because having internal logic (such as, if some quantity is positive do this, but if it isn't, do that) does not vectorize very well. In that case, the proportion of the execution time devoted to the function can rise, leading to requests for a reduction in its calculation time.

When the Cray-1 machine arrived I wrote a half-precision square root in assembler language to solve exactly that problem. The code was using square roots that didn't need to be accurate to 12 places and suddenly, calculating square roots was dominating the run time. But the half-precision version could not be easily vectorized. Bob simply did not have the skills to do similar work for his library routines.

When I talked to Bob, he said that he just wasn't going to learn how to run on the Cray; he just couldn't take another new machine. I sympathized; in fact I almost had a similar feeling about new machines toward the end of my career, so I retired precisely to avoid becoming a dinosaur. We all loved and honored Bob, and nobody was going to ask him to find another assignment. He decided on his own to retire. I always remembered what Bob said about being overwhelmed by change.

I learned that 80% of a manager's time is spent dealing with the least-productive 20% of the staff. Later I wrote an essay in *Computers in Science and Engineering*[39] about how "change" had broken the social contract with people like Bob Pexton. That situation has worsened over time. We ask people to go into very narrow specialties in a high-change environment and then treat them badly if they are less productive as they age.

Artificial Intelligence And Go

I Play Go in Livermore's Sister City, Yotsukaida, Chiba Ken, Japan

I mentioned that I learned to play the game of "Go" when I was in Alberta; it's called "Wei-chi" in Chinese. I actually studied it for years, reaching a respectable level of ability for an amateur, "3-Dan". I learned a lot of Japanese at home so that I could read Go problem-books and Go news, taking several vacations to Japan for the purpose. In the photo at the left, I am playing with the Principal of the high school in Yotsukaido, Livermore's Sister City. I was

[39] Dubois, Paul F., "Brain Cancer May Be The Least Of Our Worries," *Computers in Science and Engineering,* v. 10 #3, May/Jun 2008.

there to teach a math class and an English class as part of our Sister-City exchange, but he wanted a game first. Pro tip: even if you think you speak Japanese, you can't speak math and Japanese at the same time.

Artificial intelligence was a very highly-touted research area at one point, so much so that when nothing much came of it then, there was a certain loss of luster. Chess became the focus of a lot of research. Finally IBM's "Deep Blue" program beat Gary Kasparov at Chess in a six-game match in 1997, on its second try. This was a sensation. We Go players, however, knew that the best a computer could then do in Go was a level reached by a human in a year or so. The problem was that the chess program used a "position evaluator" (noting who is ahead in this position, largely based on material won) with a tree-pruning, look-ahead algorithm. Go has so many more elements than Chess that the tree is too big, and capture of material is often insignificant, making position evaluation difficult.

For twenty years after that, computers could only beat amateur Go players. Then, IBM's Alpha Go attacked the problem by a neural net— that is, a program that decides if a move is good or bad depending on how that kind of move has fared in the past. The program could get better just by playing itself or by absorbing the records of professional games which are published and extend back hundreds of years, as Go is the national game of Japan. Interestingly, humans learn to play Go well partly with an aesthetic approach, learning what shapes in the stones tend to have good outcomes later. A friend of mine in Alberta, Keith Taylor, explained why Go players would call a shape "beautiful". Keith said, "'Shape' is like a woman— if she treats you right, pretty soon she begins to look good to you!"

In 2015, I watched the match between Alpha Go and the Korean player Lee Sedol, eighteen-times world champion and one of the strongest players who ever lived. After winning Game 1, in Game 2, Alpha Go played a move that thunderstruck me. *Honinbo* Shusaku had, in 1846, played another amazing move in the mid-game, far out in the middle of the board, and the move is today called the "Ear-reddening move of Shusaku". A local doctor, not at all good at Go, had been watching the play. The doctor said that it was his feeling that Shusaku was winning because the ears of his opponent had turned red with shock when he saw the move. Now Alpha Go's move was simply thrilling, so unusual it could not be based on just look-ahead— indeed it was probably the winning move. I immediately felt that Alpha Go was going to win the match, and that the world had changed forever. Alpha Go won four games out of five.

That was just seven years ago, and now both Go and Chess professionals are learning new ideas from the machine. The games between the "engines" are breathtaking. We are also seeing progress in robotics and in AI that is just stunning. We now have a helicopter flying itself around on Mars. The challenge will be, what do we do as more and more people

become unemployable? Already one expert believes there soon will be few jobs for people whose IQ is one sigma below average, and that the dividing line is moving up fast. We aren't having the needed conversations about what a humane society will look like in a short time.

Clifford Traps The Congressional Aides

In 1989, Physicist Cliff Rhoades wanted to hold the *Conference on Grand Challenges in Computation.* His idea was to have a bunch of presentations and discussion panels that showcased what could be accomplished in computational sciences, as a motivation for funding more supercomputers and computational-science research.

The main way to fund such things is to motivate the Congress, and since the politicians may be rather unscientific, the people who actually make a lot of the decisions are their aides. Cliff knew, however, that said aides were always going to be on their phones or wandering off to local attractions, and he wanted to trap them as well as attract them. So he held the conference on Molokai, Hawaii. Yes, the former leper colony. There was just one hotel on the end of a sandspit. No wireless. One hamburger stand some distance away. Some very poor snorkeling. Bored to tears, the aides went to the talks.

I was on the panel called "The Future of Large-scale Scientific Computations". In a real coup, Cliff had gotten Kenneth Wilson, Nobel Prize winner in Physics in 1982, to come to the conference and be on this panel. This was the first Nobel awarded to someone that worked in computational simulation. I was grateful for his presence because something I said caused someone in the audience to jump up and say very vehemently that I was wrong. Wilson spoke up and said I was right, and so the guy had to shut up. I was often on panels because I pushed the envelope.

Sometime Secrets

Z-Division, we heard, handled intelligence estimates on foreign powers. It was so very secret that once my colleague Gary Rodrigue came back to Numerical Mathematics Group and said he'd interviewed for a job with them, but he wasn't going to take it because they could not tell him anything about the job— not what it was, nor was there any travel involved, nothing. Just "Do you want a job?"

When I had a job that required Classified material, I had a big safe to put it in. "Safe Checkers" would come by every evening to make sure I had not left it open.

From time to time, I attended Classified presentations outside my own projects. One June I went to one about one of the projects later called “Star Wars”. The first slide gave the code name of the project and said that even the code name was classified. I read the name the next December in the *San Francisco Chronicle*.

I believe that the political/finance people above us, both in the Lab, the DOE, and the Congress, cannot keep their mouths shut. The continual struggle for money rules everything. The project held the meeting because it needed people to know about itself. If for no other reason, a project’s personnel would get raises depending on the internally-perceived importance of the project, for as Dr. Strangelove said, there is no point to a Doomsday Device if you keep it a secret.

Teaching Spreads My Face

Teaching Fortran

While I was still in Numerical Mathematics Group (NMG), I decided to offer a 30-hr. videotape course on Fortran and Programming. I sat at a table in a studio, with prepared transparencies being recorded from a camera above, and there was a screen inset of me talking. It was fancy stuff for those days.

Subsequently I learned that at least two women who took the class subsequently started a career transition from clerical or publications work to applications programmer. Years later, I was walking over to Building-113 when someone coming the other way said, “Hi, Paul”. I didn’t recognize the persond, and I said, “Excuse me, I don’t remember you.” “Oh!,” he replied. “Of course you don’t. We’ve never met. I took your Fortran course, and I feel like I know you.”

Teaching OOP

Later I offered a couple of courses in person that I humorously advertised as “Dubois Free University”. You couldn’t get credit, but I would write a letter if asked, saying that you had learned the material from me. No one ever took me up on that.

I gave a course on Object-Oriented Programming, and in particular emphasized Meyer’s theory of software-enforced contracts and why that improves correctness. Most of the key people in what would come to be the Kull project got their inspiration from that course and from Basis, and when the time came, they chose Python-steered C++ for the project. Later I

joined their group. We implemented what we could of Meyer's theory in C++, and I rounded it out by supplying a regime of unit tests and integration tests.

Formal Languages and Automata

Just outside the Livermore Lab gate on the East side was a building containing a branch of U. C. Davis, which we called "Teller Tech". My NMG mentor Garry Rodrigue had become a professor there just about when I went to MFE. Because he knew me, I was asked to teach one semester there, and taught a graduate course in Formal Languages and Automata, which is perhaps one of the most "math-like" in their curriculum. The hoops one had to jump through to teach there were a deterrent, though. There were separate paychecks for it, and I was treated as a part-time employee for service credit.

Unfortunately the students were of disappointing quality. For example, one week I assigned a problem in the book as homework. When it turned out that no one had been able to solve it, I realized I'd made an error— the problem would be trivial after the next chapter but not before it. I explained my mistake, and stayed up until 3 A. M. figuring out a solution that didn't use the next chapter. I passed out my solution. I put the problem on the midterm. No one could do it even using the next chapter. I put it on the final. Crickets.

Wargames On Computers

Military simulation games had been a big hobby of mine. These "wargames" took hours or days to play. Sometimes a friend would play with me, but most of the time I played them at home, alone. My wife had to make a big heavy cover out of wood so that the cats could not disturb the game, but some armored divisions ended up with bite marks anyway. With this background I was of course excited by the first video games of any kind.

The first video game we all played was called "Pong" (1972), and it was unbelievably crude: two paddles and a "ball", a line down the middle of the "court", and two numbers at the top keeping score. The company that produced it was called Atari, which is a word Go players use as a courtesy to warn the opponent that some of his stones are liable to be captured on the next move. While most Atari-system users had a model 2600 machine, using cartridges to hold the games, Atari eventually started making an actual computer in 1979.

I first saw an Atari 800 at a fantasy convention held at a hotel in San Mateo. A group of us who played war games at the Lab at lunchtime attemded. I was mesmerized by someone demonstrating a wargame on an Atari 800 with a scrolling map. Imagine, a wargame where

the cats could not knock over the pieces! One of the members of our lunch group actually got a job at the Lab's Conflict Simulation Lab and got to do this for a living.

I had to overdo it of course, as I mentioned in the discussion about the Forth language. I bought an 800 with the maximum amount of memory, two floppy disk drives, a tape (!) drive that used cassette tapes, and a very, very slow modem. It cost about $2600 all together, a fantastic sum then. My wife never complained about this or any other of my "early-adopter" hardware purchases.

There was a cartridge for the Basic language, and a cartridge for Assembler language for the 6502 chip. My wife learned enough Basic to write a checkbook program, and then, satisfied that she knew what that was about, never touched it again.

I meantime was dying to reproduce that scrolling map. The Atari was drawing the screen top to bottom, and then entering what was called the "vertical blank". During the time it was getting ready to start drawing the screen again, the programmer had a chance to catch the vertical-blank-interrupt event and change the memory from which the screen would be drawn. I succeeded in getting that to work except that every once in a while a big glitch would happen and the screen would go crazy before going back to normal. It turned out that the 6502 had two separate arithmetic modes, binary and decimal. Once in a while the interrupt would occur when the machine was in decimal mode, unbeknownst to me, who did not even know there was such a thing. It took me months to find the problem. Nowadays, a simple search on one of the technical web sites would solve that kind of problem, but back then we were on our own.

The lunchtime group of wargamers suffered an ignominous end. After we invested nine months battling Romans playing "Siege of Jerusalem" in an unused conference room, the Lab held a "Family Day". The night before, a confused guard, seeing these hundreds of pieces and a map, apparently feared that it might be classified despite its unclassified location, so he picked it all up. We never had the heart to try again.

A CAUTION

Let me, like President Eisenhower did, conclude with a caution. The results of computational science have been so spectacular that we may be vulnerable to believing anything that a computer tells us, forgetting that behind those calculations are men and women who make programming errors; who neglect factors, believing they are not important when they are; who sometimes use algorithms outside of their range of application; or who have a vested interest in the outcome. They may lack the full set of skills needed for computational

science yet want to work alone. Confirmation bias is very powerful— as Simon and Garfunkel put it, “A man sees what he wants to see and disregards the rest.” The peer-review procedure can be corrupted by political views or personal relationships as well.

Working for the government has its own set of problems. When a government tells you their calculations show X, it isn’t necessarily so. It may be that saying X is true has some political advantage; or the people involved want it to be true or believe it to be true despite “the science”. I say this because I once told the DOE “X”, and they soon announced “not X”, I assume for political reasons. The result is that you cannot trust the government to report the work of its science correctly. Of course, they make those results a secret, so that nobody can legally tell the public otherwise. During the Covid pandemic we have seen the havoc that the resulting distrust can engender.

APPENDIX

CURRICULUM VITAE

Education

B.A. , Mathematics, University of California, Berkeley, (1966).
M.A. , Mathematics, University of California, Davis (1967).
Ph. D., Mathematics, University of California, Davis (1970).

Professional Experience

I. W. Killam Postdoctoral Fellow, University of Alberta, Edmonton, Alberta, Canada, 1970-73

Assistant Professor of Mathematics, New Mexico Highlands University, Las Vegas, N. M., 1973-76

Visiting Staff Member, Los Alamos Scientific Laboratory (Group T- 7), Los Alamos, N. M., 1974-76

Mathematician / Computer Scientist, Lawrence Livermore National Laboratory, 1976-2012

1. 2001-2012: Software Architect, Kull Project. Design and Development of a three-dimensional computational-steered physics simulation. Center For Applied Scientific Computations. Retired 2005.

2. 1997-2001: Computer Scientist, Program for Climate Model Diagnosis and Intercomparison. Development of tools for analyzing the output of computer models.

3. 1995-1997: Project Leader, X-Division. Responsible for computer science direction for X-Division. Developing next-generation systems for steerable object-oriented computations, directing computer science team.

4. 1987-1995: Chief Computer Scientist, General Physics Division. Responsible for computer science direction for Inertial Confinement Fusion. Led this team in the

complete modernization of the LASNEX program and associated tools, and ported it to workstations. Also Group Leader, ICF Group, Lasers and Energy Computer Division, Computations Applications Organization, 1993-1995.

5. 1984-1987: Project Leader, MERTH/Basis Project, Magnetic Fusion Energy Program, Theory and Computations Group. Led a project to create a comprehensive modeling code for Tandem Mirror magnetic fusion machines. Created Basis, a system for producing scientific application programs.

6. 1980-1984: Group Leader, Programmatic Applications Group, Computer and Mathematics Research and Development Division. Led a group of 7-9 mathematicians in improving the speed, accuracy, and capability of the Laboratory's large computer programs. Research and applications in radiation transport, linear algebra, vectorization. Participated in Parallel Processor Project, Ionosonde Project.

7. 1976-1980: Numerical Mathematics Section. Improved numerical techniques in large simulations. Numerical and vector algorithm research and development, Numerical Library support.

Professional Service

Editor, Scientific Programming Department, *Computers in Physics*, 1993-1998.

Editor, Scientific Programming Department, *IEEE Computers in Science and Engineering*, 1999-2005; stand-alone column "Café Dubois" 2006-2008.

Program Committee, International Conference on the Technology of Object-Oriented Languages and Systems (TOOLS Europe), 1989-2005

Program Committee, *International Python Conference,* 1997-. Hosted the Conference at Livermore National Laboratory, June, 1996.

Panel Member, National Research Council, National Academy of Sciences, Ford Graduate Fellowships for Minorities, 1991-1992.

Teaching "Formal Languages and Automata", graduate course, Department of Applied Science, University of California, Davis. 1988.

Teaching LLNL courses "Fortran" (1978), "Object-Based Fortran 95" (1998), and "Object-Oriented Analysis, Design, and Programming" (1994).

Judge, "Eiffel Struggle Programming Contest", Non-profit International Consortium for Eiffel, 1998.

Consulting

Manager and lead engineer for “EiffelMath Library”, an Eiffel interface to the NAG numerical analysis library, for Interactive Software Engineering, Goleta, CA, 1995-96.

Consultant, Righthinking, Inc., Alameda, CA. 1997.

Course Design and Delivery, U. S. Environmental Protection Agency (2000?).

Honors and Awards

Kraft Prize for Scholarship, University of California (1963).

NDEA Title IV Fellow, 1968-69.

National Science Foundation Graduate Fellow (1969-1970)

I. W. Killam Postdoctoral Fellow, University of Alberta (1970-1972).

Physics Distinguished Achievement Award, Lawrence Livermore National Laboratory, 1991.

Principal Investigator, Institutional Research and Development Grant: "MERTH: A Simulation Code for Tandem Mirror Machines," FY1985-$110,000; FY1986-$200,000; FY1987- $200,000. Physics Department, Lawrence Livermore National Laboratory.

Bruno, J. and P. F. Dubois, “Concurrent Extensions to an Object-Oriented Language”, Institute for Scientific Computation, Lawrence Livermore National Laboratory, FY 1990-91.

Cohen, Ron and Paul F. Dubois, "Disparate-Scale Fluid Phenomena", Laboratory Director's Initiative, FY1993 $750,000.

Weaver, Thomas, P. Eltgroth, and P. Dubois, “Integrated Astrophysical Modeling”, Lawrence Livermore National Laboratory, FY1995, $250,000.

Richard London, “LATIS3D, Laser-Tissue-Interaction Modeling”, FY1999. $300,000. Co-investigator.

Invited Addresses

1. "A New Architecture for Large Scientific Simulations," NSF and NASA Workshop on Parallel Computations in Heat Transfer and Fluid Flows (1984).

2. “Basis: Setting The Scientist Free”, 16th IEEE Conference on Numerical Simulation of Plasmas**,** Buffalo, New York, (1989).
3. “The Future of Large-scale Scientific Computations” (Panel), Conference on Grand Challenges in Computation**,** Molokai, HI (1989).
4. “The Evolution of Large-scale Scientific Computations”, Keynote Address, Fifth International Eiffel User’s Group Meeting**,** The Eiffel Tower**,** Paris, France, (1990).
5. “Steering Object-Oriented Scientific Calculations”, Computational Accelerator Physics 96, Williamsburg, VA, 1996.

Books

P. F. Dubois, Object Technology for Scientific Computing, Prentice-Hall, NJ, 1997. 288 pp.

Refereed and Invited Publications

1. D. O. Cutler and Paul F. Dubois, "A Generalization of Final Rank for P-Primary Albelian Groups," Canadian J. Math., 22, 1118 (1970).
2. D. O. Cutler and Paul F. Dubois, "Generalized Final Rank for Arbitrary Limit Ordinals," Pacific J. Math., 37, no. 2 (1971), pp. 345-352.
3. Dubois, P. F., "Generally pα-torsion Complete Abelian Groups," Transactions, Amer. Math. Soc., 159 (1971), pp. 245-255.
4. P. F. Dubois and S. K. Sehgal, "Another Proof of the Invariance of Ulm's Functions in Commutative Modular Group Rings," Math. J. Okayama Univ., 15, #2 (1972).
5. P. F. Dubois and G. H. Rodrigue, "Operator splitting on Vector processors," in Advances in Computer Methods for Partial Differential Equations - II, (IMACS (AICA), New Brunswick, NJ, 1977), pp.195-197.
6. P. F. Dubois and G. H. Rodrigue, "An Analysis of the Recursive Doubling Algorithm," in High Speed Computer and Algorithm Organization (Kuck, et al., Eds., Academic Press, New York, 1977).
7. P. F. Dubois, A. Greenbaum, and G. H. Rodrigue, "Approximating the Inverse of a Matrix for Use in Iterative Algorithms on Vector Processors," Computing 22, pp. 257-268 (1979).

8. P. F. Dubois and P. L. Dubois, "Measuring Dissent Behavior on State Courts: An Application and an Adaptation of Known Measurement Techniques," Polity, XIII (1980) pp. 147-158.
9. P. F. Dubois and T. S. Axelrod, "Multigroup Diffusion with a General ODE Solver", in Proceedings of the Nuclear Explosives Code Development Conference-1980 (U), p 74- 74-78. (SRD)
10. "Swimming Upstream: Calculating Table Look-ups and Piecewise Functions," Parallel Computation (G. Rodrique, Ed., Academic Press, New York, 1982).
11. T. S. Axelrod, P. F. Dubois, and P. Eltgroth, "A Simulator for MIMD Performance Prediction— Application to the S-1 MkIIa Multiprocessor", Parallel Computing (1984) p. 237-274.
12. T. S. Axelrod, P. F. Dubois, and C. E. Rhoades, Jr., "An Implicit Scheme for Calculating Time and Frequency Dependent Radiation Transport in One Dimension" J. Comp. Phys. 54, no. 2, p. 205ff (1984).
13. W. M. Nevins, P. F. Dubois, J. M. Gilmore, G. R. Smith, T. B. Kaiser, R. P. Freis, and B. I. Cohen, "Merth— A Modeling Code for Tandem Mirrors," in 11th International Conference on Numerical Simulation of Plasmas**,** Montreal, Canada, June 25-27, 1985.
14. W. M. Nevins, B. Boghosian, R. H. Cohen, W. F. Cummins Jr., P. F. Dubois, A. Friedman, L. L. LoDestro, Y Matsuda, L. D. Pearlstein, G. D. Porter, M. E. Rensink, T. D. Rognlien, G. R. Smith, J. J.Stewart, and M. Phillips, "A Tandem Mirror Modeling Code," in Plasma Physics and Controlled Nuclear Fusion Research, 1986, Vol 2. International Atomic Energy Agency, Vienna, 1987.
15. P. F. Dubois, “LASNEX Under Basis”(U), in Proceedings of the Nuclear Explosives Code Development Conference-1990 (U), (SRD)
16. P. F. Dubois, "Object-Oriented Programming", in Computers in Physics, Nov. 1991.
17. P. F. Dubois, "Try Something New: Object-Oriented Thinking", Guest Editor's Essay, in Computers in Physics, v.6, # 5, Sep/Oct 1992.
18. T.-Y. Yang, G. Furnish, and P. F. Dubois, “Steering Object-Oriented Scientific Calculations”, Technology of Object-Oriented Languages and Systems Conference, August 1997, Santa Barbara, CA.
19. P. F. Dubois, “Scientific Components Are Coming”, Computer, v. 32, #3, March 1999.

Scientific Programming Department

The Scientific Programming Department is a column that appeared six times a year. It began in 1993 in *Computers in Physics*, and continued after a journal merger in *IEEE Computing in Science & Engineering.* As the Department Editor, I chose and edited each article. This is a list of the articles authored or co-authored by me, and also the smaller articles I wrote to accompany the main article. Starting in September, 2006, my column was a separate article. Many of these are available in the ACM Digital Library.

1. Lee Busby and P. F. Dubois, "Powerful, Portable Fortran Programming", Computers in Physics, v.7, #1, Jan/Feb 1993.
2. Lee Busby and P. F. Dubois, "Portable Programming and the Fortran Standard", Computers in Physics, v. 7, #2, Mar/Apr 1993.
3. P. F. Dubois, "Perl by Example", in Computers in Physics, v. 7, #5, Sep/Oct 1993.
4. P. F. Dubois, "Making Applications Programmable", in Computers in Physics, v.8, #1, Jan/Feb 1994.
5. S. A. Brown, P.F. Dubois, and D. Munro, “Writing and Using PDB Files”, in Computers in Physics**,** v. 9, #2. March/April 1995.
6. P. F. Dubois, K. Hinsen, and J. Hugunin, “Numerical Python”, Computers in Physics, v. 10, #3, May/June 1996.
7. P. F. Dubois and T.-Y. Yang, “Extending Python”, Computers in Physics, v. 10, #4, July/August 1996.
8. P. F. Dubois, “The Future of Scientific Programming”, Computers in Physics, v. 11, #2, Mar/April 1997.
9. P. F. Dubois, “Is Java for Scientific Programming?”, Computers in Physics, v. 11, #6, Nov/Dec 1997.
10. Holmes, Lewis, and Department Editors, “The Future of Scientific Computing”, Computers in Physics, v. 11, #6, Nov/Dec 1997.
11. P. F. Dubois, “Ten Good Practices in Scientific Programming”, Computing in Science & Engineering, v1. #1, Jan/Feb. 1999.
12. P. F. Dubois, “Extending Maple with Compiled Routines”, Computing in Science & Engineering, v. 2, #4 Jul/Aug 2000.

13. Scherer, David, Sherwood, Bruce, and P. F. Dubois, “VPython: 3D interactive scientific graphics for students”, Computing in Science & Engineering, v. 2, #5, Sept/Oct 2000.
14. P. F. Dubois, “Fortran: A Space Odyssey”, Computing in Science & Engineering, v. 3, #2, March/Apr 2001.
15. P. F. Dubois, “Designing Scientific Components”, Computing in Science & Engineering, v. 4, #5 Sep/Oct 2002
16. P. F. Dubois, “Why Johnny Can’t Build”, Computing in Science & Engineering, v. 5, #5, Sep/Oct 2003.
17. Johnson, Jeffrey N. and P. F. Dubois, “Issue Tracking”, Computing in Science & Engineering, v. 5, #6, Nov/Dec 2003
18. Downing, Glen, Cottom, Teresa, and P. F. Dubois, “Data Sharing in Scientific Simulations”, Computing in Science & Engineering, v. 6, #3, May/Jun 2004.
19. P. F. Dubois, “Cafe Dubois, Polling Place Edition”, Computing in Science & Engineering, v. 6, #6, Nov/Dec 2004.
20. P. F. Dubois, “Maintaining Correctness in Scientific Programs”, Computing in Science & Engineering, v. 7, #3, May/June 2005.
21. P. F. Dubois, “Bean-Counted Research is Smelly”, Computing in Science & Engineering, v. 7, #6, Nov/Dec 2005.
22. P. F. Dubois, “A Nest of Pythons”, Computing in Science & Engineering, v. 7, #6, Nov/Dec 2005.
23. P. F. Dubois, “Google Summer of Code / Numeric Python”, v.8, #1, Jan/Feb 2006.
24. P. F. Dubois, “In The Senior League”, Computing in Science & Engineering, v. 8, #2, Mar/Apr 2006
25. P. F. Dubois, “He’s Baaaack!”, Computing in Science & Engineering, v. 8, #5, Sep/Oct 2006.
26. P. F. Dubois, “So Sue Me”, Computing in Science & Engineering, v. 8, #6, Nov/Dec 2006.
27. P. F. Dubois, “Djange Me”, Computing in Science & Engineering, v. 9, #1, Jan/Feb 2007.

28. P. F. Dubois, “The Future That Never Was”, Computing in Science & Engineering, v. 9, #2, Mar/Apr 2007.
29. P. F. Dubois, “Guest Editor’s Introduction: Python: Batteries Included, Computing in Science & Engineering, v. 9, #3, May/Jun 2007
30. P. F. Dubois, “Customer Service Says The Darnedest Things”, Computing in Science & Engineering, v. 9, #4, Jul/Aug 2007
31. P. F. Dubois, “Into The Future”, Computing in Science & Engineering, v. 9, #5, Sep/Oct 2007.
32. P. F. Dubois, “Career Contradictions”, Computing in Science & Engineering, v. 9, #6, Nov/Dec 2007.
33. P. F. Dubois, “Sprinting Ain’t Easy”, Computing in Science & Engineering, v. 10 #1, Jan/Feb 2008.
34. P. F. Dubois, “Brain Cancer May Be The Least of Our Worries”, Computing in Science & Engineering, v. 10 #3, May/Jun 2008.
35. P. F. Dubois, “Goodbye to All That”, Computing in Science & Engineering, v. 10, #4, Jul/Aug 2008.
36. P. F. Dubois, “Testing Scientific Programs”, Computing in Science & Engineering, v. 14, #4, Jul/Aug 2012.
37. P. F. Dubois, “Aging Swiftly”, Computing in Science & Engineering, v. 17, #5, Sept/Oct 2015.

Other Publications

1. Leontief Substitution Models with Constraints, Los Alamos Scientific Laboratory, Los Alamos, NM, LA-UR-75-1583 (1975).
2. PRETART/TARTV User's Manual, Lawrence Livermore National Laboratory, Livermore, CA, UCID-17518 (1977).
3. Volume Calculation and Geometry Checking in a Monte Carlo Transport Code, Lawrence Livermore National Laboratory, Livermore, CA, UCID-17522 (1977).
4. P. F. Dubois and G. H. Rodrigue, Operator Splitting on the STAR Without Transposing, Lawrence Livermore National Laboratory, Livermore, CA, UCID-17515 (1977).

5. D. B. Morris, A. C. Hindmarsh, and P. F. Dubois, GEARV and GEARST: Vectorized Ordinary Differential Equation Solvers for the 7600 and STAR Computers, Lawrence Livermore National Laboratory, Livermore, CA, UCID-30119, Rev. 1 (1977).
6. P. F. Dubois, S. I. Warshaw, and M. Keepler, On the Decomposition of Crossed Polygons Into Simple Ones, Lawrence Livermore National Laboratory, Livermore, CA UCID-17741 (1978).
7. P. F. Dubois and J. R. Kohn, Equation of State Table Look-up: A Case Study in Vectorization, Lawrence Livermore National Laboratory, Livermore, CA, UCRL-81184 (1978).
8. EZVECS: Macros for Vector Programs, Lawrence Livermore National Laboratory, Livermore, CA, UCID-17927 (1978).
9. Inverse Equation of State Look-up, Lawrence Livermore National Laboratory, Livermore, CA, UCRL-52563 (1978).
10. A. C. Hindmarsh, L. J. Sloan, and P. F. Dubois, DEC/SOL: Solution of Dense Systems of Linear Algebraic Equations, Lawrence Livermore National Laboratory, Livermore, CA, UCID-30137, Rev. 1 (1978).
11. P. F. Dubois and S. I. Warshaw, Implementation of the Inverse EOS Look-up with Kohn's EOS4 System, Lawrence Livermore National Laboratory, Livermore, CA, UCIR-1328 (1979)
12. P. F. Dubois and S. I. Warshaw, Calculation of Infrasonic Pulses from a Contained Underground Nuclear Explosion to the Ionsphere (U), Lawrence Livermore National Laboratory, Livermore, CA, UCID-18335 (1979) (CNSI).
13. P. F. Dubois and S. I. Warshaw, Calculating Vertical Sounder Response to Ionspheric Disturbance Caused by an Underground Nuclear Test, 35mm movie, color, silent 5 min. Lawrence Livermore National Laboratory, Livermore, CA, TP #1372 (1980).
14. P. F. Dubois, Forward and Inverse EOS4 Package Manual, Lawrence Livermore National Laboratory, Livermore, CA, UCIR-1436 Rev. 2, (1982).
15. P. F. Dubois, MMG— A Simple Memory Manager for FORTLIB, Lawrence Livermore National Laboratory, Livermore, CA, UCID-30191 (1982).
16. P. F. Dubois and S. I. Warshaw, Preliminary Theoretical Acoustic and RF Sounding Calculations for MILL RACE, Lawrence Livermore National Laboratory, Livermore, CA, UCID-19231 (1982).

17. MERTH Contributor's Manual, Lawrence Livermore National Laboratory, Livermore, CA, (1984).
18. P. F. Dubois, "A New Architecture for Large Scientific Simulations," in Proceedings of the NSF and NASA Workshop on Parallel Computations in Heat Transfer and Fluid Flows, Dept. of Mechanical Engineering, Univ. of Maryland (1984), p74-76.
19. MERTH User's Guide, Lawrence Livermore National Laboratory, Livermore, CA, (1985).
20. P. F. Dubois, et al., "MERTH - A Comprehensive Modeling Code for Tandem Mirrors", Mirror Theory Monthly, Lawrence Livermore National Laboratory, Livermore, CA (1985).
21. "MERTH: A Simulation Code for Tandem Mirror Machines", in Institutional Research and Development FY85, Lawrence Livermore National Laboratory, Livermore, CA, UCRL-53689-85 (1986).
22. MPPL User's Manual, Lawrence Livermore National Laboratory, Livermore, CA, M-187 (1986).
23. Writing Basis Packages, Lawrence Livermore National Laboratory, Livermore, CA, M- 194, (1986).
24. P. F. Dubois and Z. C. Motteler, Basis User's Manual, Lawrence Livermore National Laboratory, Livermore, CA, M-189 (1986).
25. MPPL User's Manual, Lawrence Livermore National Laboratory, Livermore, CA, M-187, Rev. 1 (1987).
26. R. L. Reid, W. L. Barr, C. G. Bathke, J. N. Brooks, R. H. Bulmer, A. Busigin, P. F. Dubois, M. E. Fenstermacher, J. Fink, P. A. Finn, J. D. Galambos, Y. Gohar, G. E. Gorker, J. R. Haines, A. M. Hassanein, D. R. Hicks, S. K. Ho, S. S. Kalsi, K. M. Kalyanam, J. A. Kerns, J. D. Lee, J. R. Miller, R. L. Miller, J. O. Myall, Y-K. M. Peng, L. J. Perkins, P. T. Spampinato, D. J. Strickler, S. L. Thomson, C. E. Wagner, and R. S. Williams, ETR/ ITER Systems Code, Oak Ridge National Laboratory, Oak Ridge, TN, ORNL/FEDC-87- 7 (1988).
27. P. F. Dubois, et al., The Basis System, M-225, Lawrence Livermore National Laboratory, Livermore, CA (1988).
28. P. F. Dubois and Z. C. Motteler, Basis: Setting the Scientist Free, Lawrence Livermore National Laboratory, Livermore, CA, UCRL-99889 (1988).

29. P. F. Dubois, et al., Proceedings of the First Basis User's Group Meeting, Lawrence Livermore National Laboratory, Livermore, CA, M-4690 (1988).

30. P . F. Dubois, Basis/Lasnex User’ s Manual, Lawrence Livermore National Laboratory, Livermore, CA (1990).

31. R. H. Cohen, B. I. Cohen, and P. F. Dubois, Comprehensive Numerical Modeling of Tokamaks, UCRL-ID-105650, Lawrence Livermore National Laboratory, Livermore, CA (1991) 15pp.

32. Dubois, Paul F. and Dan M. Nessett, Scientific Distributed Computing in the DOE: Issues and Approaches, UCRL-ID-1065333, Lawrence Livermore National Laboratory, Livermore, CA (1991) 17pp.

33. L. Busby, P. F. Dubois, and S. Wilson, "Lasnex Evolves to Exploit Computer Industry Advances", in Inertial Confinement Fusion Quarterly Report**,** Oct-Dec 1992, v. 3 #1. p. 50-53. Lawrence Livermore National Laboratory, UCRL-LR-105821-93-1 (1993).

34. P. F. Dubois et al., The Basis System**,** Parts I-VI, Lawrence Livermore National Laboratory, UCRL-MA-118543, 1995.

35. P. F. Dubois, “Python Extensions Documentation”, UCRL-128569, Parts I-III, 1997.