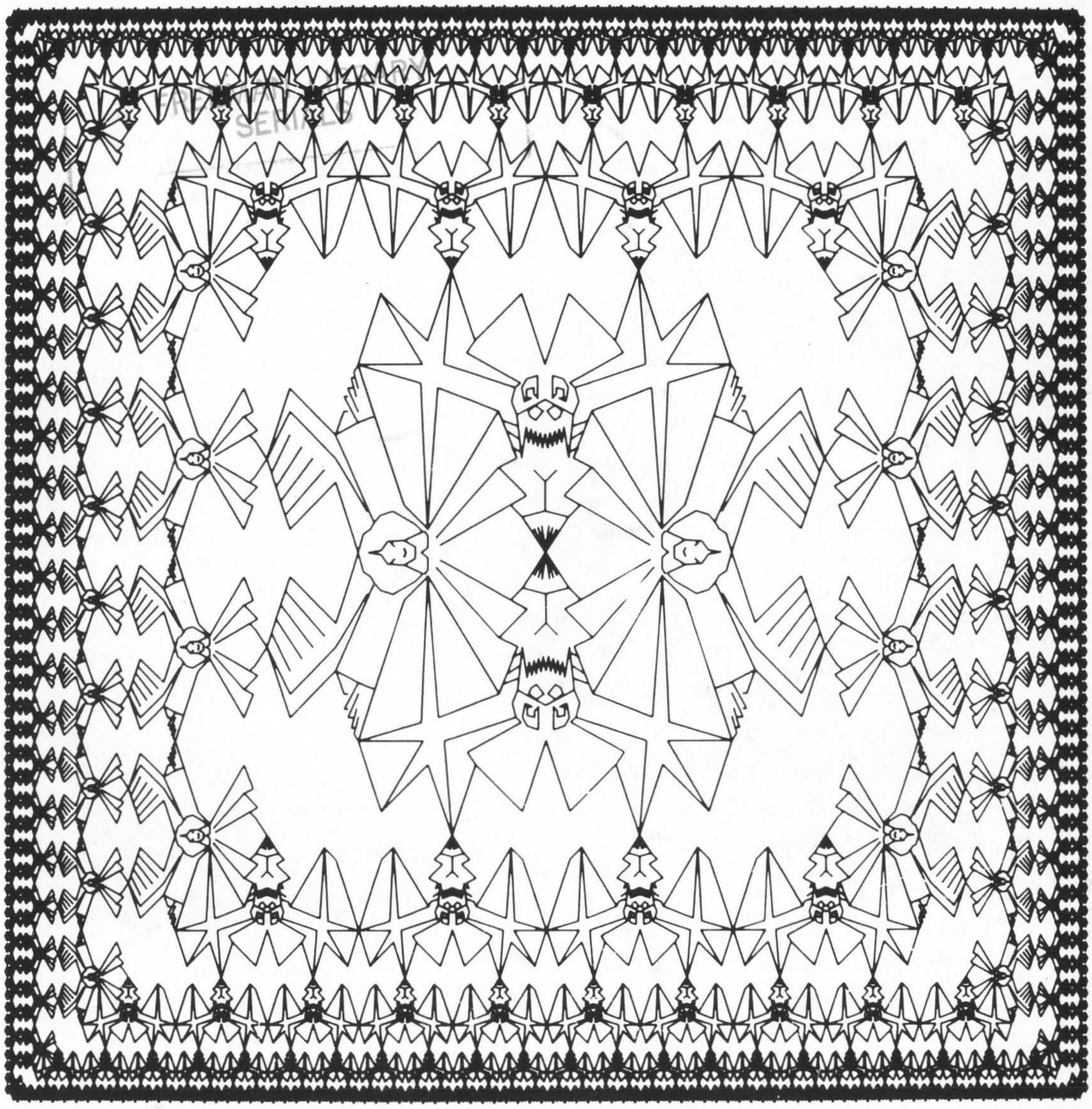
computers and people

formerly *Computers and Automation*

July—August, 1987
Vol. 36, Nos. 7-8



STRUGGLE BETWEEN GOOD AND EVIL *by William J. Kolomyjec*

The Computer Almanac and Computer Book of Lists — Instalment 54

Neil Macdonald, Assistant Editor

26 APHORISMS (List 870701)

To do two things at once is to do neither.

A good reputation is more valuable than money.

It is good to moor your boat with two anchors.

Many receive advice, few profit by it.

You should hammer your iron when it is glowing hot.

When Fortune flatters, she does it to betray.

Any one can steer the boat when the sea is calm.

Practice is the best of all instructors.

It is a bad plan that admits of no modification.

Never promise more than you can perform.

Necessity knows no law except to conquer.

Nothing can be done both hastily and prudently.

Only the ignorant despise education.

Do not turn back when you are just at the goal.

Not every question deserves an answer.

He bids fair to grow wise who has discovered he is not so wise.

Familiarity breeds contempt.

You should go to a pear-tree for pears, not to an elm.

In every enterprise, consider where you would come out.

It does not matter what you are thought to be but what you are.

No one knows what he can do until he tries.

The next day is never so good as the day before.

It does not matter how long you live but how well.

It is better to learn late than never.

Prosperity makes friends, adversity tests them.

Whom Fortune wishes to destroy, she first makes mad.

(Source: some of the maxims of Publius Syrus, 42 B.C.)

65 COMPLETE UTTERANCES OF ONE WORD (List 870702)

ah
aha
aw
ay
aye

bah
bo
boo

damn
danger
darn
detour

eh
enough
er

god
good
goodbye
gosh
great

ha
hell
hello
help
here
hey
hi
hm
ho
hurray
hush

listen
lo
look

ma
ma'am
maybe
more

never
no

oh
ok
ouch
out
ow

pa
pardon
perhaps
please

rah

safe
sh
sir
slow
sorry
stop

thanks

ugh
um
unhunh

whoa
wow

yea
yes
yo

(Source: Neil Macdonald's notes. The syntactic, semantic, and pragmatic analysis of

(please turn to page 27)

Computing and Data Processing Newsletter

AMONG THE IMPOSSIBILITIES OF SDI (STRATEGIC DEFENSE INITIATIVE)

Fred Kaplan
"Boston Globe"
Boston, MA 02107
(Based on a report in the "Boston Globe" for April 24, 1987)

The "Strategic Defense Initiative" (widely known as "Star Wars" or SDI) of the Reagan administration was dealt a devastating blow in late April in a report by the American Physical Society, the nation's leading organization of physicists.

The 424-page report, highly detailed and technical, is the first assessment of SDI by scientists who have taken no apparent political stance on the nuclear arms race. In fact, the 17-member panel that wrote the report contained five scientists from four US weapons laboratories, including MIT's Lincoln Lab.

Yet the panel, which met over an 18-month period and received classified briefings from the Pentagon's SDI office, concluded -- no less pessimistically than analyses by many antiwar organizations -- that critical elements of an SDI system will not be feasible until at least the year 2000 and that an effective, full-blown SDI system may not be feasible at all.

Sen. William Proxmire (D-Wis.), a leading SDI critic, said yesterday, "The most impressive aspect of this is the remarkable balance of the panel. These are people who have no axes to grind."

Sen. John Kerry, another SDI opponent, said the report marks "further evidence that the Reagan administration's more interested in rushing ahead with some kind of SDI deployments than it is in hard science or sound defenses. I suspect the report will be a significant factor in raising skepticism as Congress considers the SDI budget."

President Reagan is asking $5.6 billion for SDI in fiscal year 1988, and has said he wants to deploy a partial space-based SDI system by the mid-1990s.

A spokesman with the SDI office, Lt. Col. Terry Monrad, said of the report, reading from a prepared statement: "We find the conclusions to be subjective and unduly pessimistic. ... The report was a snapshot in time that dates to the preparation of the report. We have made significant progress in the intervening period."

However, the report's contents cast doubt on the relevance of Monrad's reply. The study was finished and submitted to the SDI office for a security review in September 1986. The study concludes that nearly every aspect of SDI technology concerned with lasers, particle beams and space-based power supplies will require improvements by a factor of 10, to 100 or even 1000 or more before any part of a system can work.

Not even the cheeriest SDI optimist claims improvements of that scale have been made since September or, for that matter, since the SDI program began four years ago.

The report concludes:

- All kinds of lasers and particle beams, based in space or on the ground, will require imporvements by a factor of at least 100 in power output and beam-refinement "before they may be seriously considered" anti-missile weapons.

- The optics for the telescopes needed to locate and track enemy missiles as they dart through space must also be improved by at least 100 times.

- Even if this technology were developed and fitted into a weapons system, the Soviets could either destroy it with their own laser weapons or outflank it with decoys that may be much less difficult and expensive to produce.

(please turn to page 27)

Vol. 36, Nos. 7-8
July-August, 1987

computers and people

formerly *Computers and Automation*

"Computers and People" (ISSN 0361-1442), formerly "Computers and Automation," is published every two months at 815 Washington St., Newtonville, MA 02160 U.S.A., by Berkeley Enterprises, Inc. Printed in U.S.A. Second-class postage paid at Boston, MA and additional mailing points.

Subscription rates, effective Sept. 1, 1986: U.S.A., $22.00 for one year, $43.00 for two years; elsewhere, add $7.00 per year.

NOTE: The above rates do not include our publication, the "Computer Directory and Buyers' Guide." To receive this, please add $20.00 per year to your subscription rate in the U.S.A., and $23.00 per year elsewhere.

NOTE: No organization in Switzerland or Morocco is authorized or permitted by us to solicit for, or receive payment for, "Computers and People" or the "Computer Directory and Buyers' Guide." All subscriptions to and payments for these publications should be sent directly to Berkeley Enterprises, Inc.

Please address all mail to: Berkeley Enterprises, Inc., 815 Washington St., Newtonville, MA 02160, U.S.A.

Postmaster: Please send all forms 3579 to Berkeley Enterprises, Inc., 815 Washington St., Newtonville, MA 02160, U.S.A.

Change of address: If your address changes, please send us both your new address and your old address (as it appears on the magazine address imprint), and allow four weeks for the change to be made.

The magazine of the design, applications, and implications of information processing systems — and the pursuit of truth in input, output, and processing, for the benefit of people.

Announcement

The Computer Directory and Buyers' Guide is still being updated in our computer data base for the next *Directory* edition. We hope we will have this, the 28th edition, ready soon for mailing to subscribers.

Front Cover Picture

The front cover shows a sample of art by William J. Kolomyjec. Using the forms of angels and devils to represent the ideas of good and evil, he creates a mosaic-like design that is interesting to study. The computer is used to execute this graphic that is difficult, if not impossible, to do by hand.

Computer Field → Zero

There will be zero computer field and zero people if the nuclear holocaust and nuclear winter occur. Every city in the United States and the Soviet Union is a multiply computerized target. Radiation, firestorms, soot, darkness, freezing, starvation, megadeaths, lie ahead.

Thought, discussion, and action to prevent this earth-transforming disaster is imperative. Learning to live together is the biggest variable for a computer field future.

Signals in Table of Contents

[A]	–	Article
[C]	–	Monthly Column
[E]	–	Editorial
[EN]	–	Editorial Note
[O]	–	Opinion
[FC]	–	Front Cover
[N]	–	Newsletter
[R]	–	Reference

Type of Subscription

*D ON YOUR ADDRESS IMPRINT MEANS THAT YOUR SUBSCRIPTION INCLUDES THE COMPUTER DIRECTORY AND BUYERS' GUIDE. *N MEANS THAT YOUR PRESENT SUBSCRIPTION DOES NOT INCLUDE THE COMPUTER DIRECTORY.

Editorial

Progress That Is Not Possible

Edmund C. Berkeley, Editor

"The difficult we do right away; the impossible takes a little longer."

This is a favorite saying of many organizations which have big tasks to do and much eagerness to do them. It is a natural and comforting exaggeration of one's power and success to date.

But the implied promises to accomplish the impossible are regularly false. Over and over again such promises are doomed to be lies. The persons who believe them are doomed to error plus the damages and the costs of error.

Perhaps the largest class of what we may call "impossible progress" is the rather human belief that a great advance of human capacity to make progress in some direction will continue in the future unchanged.

For example, for some 40 years the historical progress in sequential computing power has been close to the change from 3 additions per second to close to 10 billion (10 to the 10th power) additions per second. But it is already clear in 1987 that in the next 40 years progress in sequential computing power that reaches to 100 billion billion (10 to the 20th power) additions per second is not physically attainable. One barrier is the finite speed of light.

For another example, consider the increase of human population from about 2 billion in 1947 to about 5 billion in 1987. But unlimited doubling of human population every 40 years cannot be accomplished on a limited, finite earth. Hundreds of obstructions lie in the path. Suppose we estimate that a plot of fertile land 100 feet by 100 feet is sufficient to support one human being.

Then:

(the surface area of the earth) TIMES

(1/4 land, not water) TIMES

(7/10, fertile, not infertile) DIVIDED BY

(area of plot, estimated for one human)

EQUALS (estimated maximum human population of the earth)

Calculating, with due attention to dimensions and units, we find the result is 90 billion persons.

How soon is this phenomenon to happen? If 2 billion persons in 1947 rises to 5 billion persons in 40 years to 1987, then we project 10 billion in 2027, and so on, to 80 billion in the middle of the 2100s. A very large change of human views must happen as we approach "standing room only". Will that change happen?

Perhaps the next largest class of "progress that is not possible" comes from the ideas and fantasies of clever, informed, and persuasive people. Such people can be found in many sections of society, particularly, government, politics, and big business. They spread their point of view through publicity and propaganda. Often they restrict or deflect what is said in newspapers, radio, and television, so that the media of a country do not report the whole truth, and so ordinary people make biased and wrong judgements. The conditions of a country at one time become substantially changed at a later time by special interests who have control over the information which ordinary people can find out. A way of expressing this point of view is "What is good for General Computers is what is good for the country."

Unfortunately, what is good for the progress of General Computers is not necessarily what is good for the progress of the country. A country contains many kinds of interests and occupations, from medical investigations to the growth of food, from nursing to farming. In modern industrial countries there may be over a thousand occupations. Each one relies on an economic need which the persons in that occupation work to satisfy. General Computers on television spreads its appeal to buy computers. But there are many other appeals to buy other goods.

Intense competition, major changes in the technology of producing goods, inadequate training for working in new activities, and many other factors interfere, and together lead to much "progress that is not possible." And computers could be used to solve many of the mysterious failures. But who would pay for the work and who would spread the conclusions? Why not General Computers? Ω

Back-of-the-Envelope Estimating

Dr. Jon Bentley
Computing Science Research Center
AT&T Bell Laboratories
Murray Hill, NJ

Early in the life of a system, rapid calculations can steer a system designer to make a rational choice between two appealing alternatives.

The Outflow of the Mississippi River

In the middle of a fascinating conversation on software engineering Bob Martin asked me, "How much water flows out of the Mississippi River in a day?" Because I had found his comments up to that point deeply insightful, I politely stifled my true response and said, "Pardon me?" When he asked again I realized that I had no choice but to humor the poor fellow, who had obviously cracked under the pressures of running a large software shop within Bell Labs.

My response went something like this. I figured that near its mouth the river was about a mile wide and maybe twenty feet deep (or about one two-hundred-and-fiftieth of a mile). I guessed that the rate of flow was five miles an hour, or a hundred and twenty miles per day. Multiplying

$$1 \text{ mile} \times 1/250 \text{ mile} \times 120 \text{ miles/day} \approx 1/2 \text{ mile}^3\text{/day}$$

showed that the river discharged about half a cubic mile of water per day, to within an order of magnitude. But so what?

At that point Martin picked up from his desk a proposal for the computer-based mail system that AT&T developed for the 1984 Summer Olympic games, and went through a similar sequence of calculations. Although his numbers were straight from the proposal and therefore more precise, the calculations were just as simple and much more revealing. They showed that, under generous assumptions, the proposed system could work only if there were at least a hundred and twenty seconds in each minute. He had sent the design back to the drawing board the previous day. The conversation took place in early 1983; but the final system was used during the Olympics without a hitch.

The Engineering Technique of Estimating

That was Bob Martin's wonderful if eccentric way of introducing the engineering technique of "back-of-the-envelope" calculations. The idea is standard fare in engineering schools and is bread and butter for most practicing engineers. Unfortunately, it is too often neglected in computing.

These basic reminders can be quite helpful in making back-of-the-envelope calculations.

Two Answers Are Better Than One

When I asked Peter Weinberger how much water flows out of the Mississippi per day, he responded, "As much as flows in." He then estimated that the Mississippi basin was about 1000 by 1000 miles, and that the annual runoff from rainfall there was about one foot (or one five-thousandth of a mile). That gives

$$1000 \text{ miles} \times 1000 \text{ miles} \times 1/5000 \text{ mile/year} \; / \; 400 \text{ days/year} \approx 1/2 \text{ mile}^3\text{/day}$$

or a little more than half a cubic mile per day. It's important to double check all calculations, and especially so for quick ones.

As a cheating triple check, an almanac reported that the river's discharge is 640,000 cubic feet per second. Working from that gives

$$640{,}000 \text{ ft}^3\text{/sec} \times 3600 \text{ secs/hr} \times 24 \text{ hrs/day} \; / \; (5000 \text{ ft/mile})^3 \approx 1/2 \text{ mile}^3\text{/day}$$

The proximity of the two estimates to one another, and especially to the almanac's answer, is a fine example of sheer dumb luck.

Quick Efficient Checks

Polya devotes three pages of his "How To Solve It" to "Test by Dimension", which he describes as a "well-known, quick and efficient means to check geometrical or physical formulas." The first rule is that the dimensions in a sum must be the same, which is in turn the dimension of the sum -- you can add feet together to get feet, but you can't add seconds to pounds. The second rule is that the dimension of a product is the product of the dimensions. The examples above obey both rules; multiplying

$$\text{miles x miles x miles/day} = \text{miles}^3\text{/day}$$

has the right form, apart from any constants.

Common Sense

Above all, don't forget common sense: be suspicious of any calculations that show that the Mississippi River discharges 100 gallons of water per day.

A few envelopes' worth of arithmetic might enable a system designer to make a rational choice between two appealing alternatives. That is a fundamentally different use than Martin's calculation for the Olympic mail system: his analysis of a single design uncovered a fatal flaw. In both cases, a short sequence of calculations was sufficient to answer the question at hand; additional figuring would have shed little light.

Early in the life of a system, rapid calculations can steer the designer away from dangerous waters into safe passages. And if you don't use them early, they may show in retrospect that a project was doomed to failure. The calculations are often trivial, employing no more than high school mathematics. The hard part is remembering to use them soon enough.

The output of any calculation is only as good as its input. With good data, simple calculations can yield accurate answers which are sometimes quite useful. In 1969 Don Knuth wrote a disk sorting package, only to find that it took twice the time predicted by his calculations. Diligent checking uncovered the flaw: due to a software bug, the system's one-year old disks had run at only half their advertised speed for their entire lives. When the bug was fixed, the sorting package behaved as predicted and every other disk-bound program also ran faster.

Often, though, sloppy input is enough to get into the right ballpark. If you guess about twenty percent here and fifty percent there and still find that a design is a hundred times above or below specification, additional accuracy isn't needed. But before placing too much confidence in a twenty percent margin of error, consider Vic Vyssotsky's advice from a talk he has given on several occasions.

The Tacoma Narrows Bridge

"Most of you," says Vyssotsky, "probably recall pictures of 'Galloping Gertie', the Tacoma Narrows bridge which tore itself apart in a windstorm in 1940. Well, suspension bridges had been ripping themselves apart that way for eighty years or so before Galloping Gertie. It's an aerodynamic lift phenomenon, and to do a proper engineering calculation of the forces, which involve drastic nonlinearities, you have to use the mathematics and concepts of Kolmogorov to model the eddy spectrum. Nobody really knew how to do this correctly in detail until the 1950's or thereabouts. So, why hasn't the Brooklyn Bridge torn itself apart, like Galloping Gertie?

The Brooklyn Bridge

"It's because John Roebling had sense enough to know that he didn't know. His notes and letters on the design of the Brooklyn Bridge still exist, and they are a fascinating example of a good engineer recognizing the limits of his knowledge. He knew about aerodynamic lift on suspension bridges; he had watched it. And he knew he didn't know enough to model it. So he designed the stiffness of the truss on the Brooklyn Bridge roadway to be six times what a normal calculation based on known static and dynamic loads would have called for. And, he specified a network of diagonal stays running down to the roadway, to stiffen the entire bridge structure. Go look at those sometime; they are almost unique.

"When Roebling was asked whether his proposed bridge wouldn't collapse like so many others, he said, 'No, because I designed it six times as strong as it needs to be, to prevent that from happening.'

"Roebling was a good engineer, and he built a good bridge, by employing a huge safety factor to compensate for his ignorance.

Do we do that? I submit to you that in calculating performance of our real-time software systems we ought to derate them by a factor of two, or four, or six, to compensate for our ignorance. In making reliability/availability commitments, we ought to stay back from the objectives we think we can meet by a factor of ten, to compensate for our ignorance. In estimating size and cost and schedule, we should be conservative by a factor of two or four to compensate for our ignorance. We should design the way John Roebling did, and not the way his contemporaries did -- so far as I know, none of the suspension bridges built by Roebling's contemporaries in the United States still stands, and a quarter of all the bridges of any type built in the U.S. in the 1870s collapsed within ten years of their construction.

"Are we engineers, like John Roebling? I wonder."

A 1982 Example

To make the above points more concrete, I'll describe how I (almost) used them in a system I built for a small company in early 1982.

The system prepared several reports a day to summarize the data on one thousand eighty-column records; the reports were each about eighty pages long. The system's predecessor ran on a large mainframe; my task was to implement a similar system on a personal computer, using interpreted BASIC.

Early in the design of the system I did simple calculations to make sure that the personal computer was up to this application. The space analysis was simple: I calculated the size of the several largest tables and found that they used only half of the 48K bytes of the machine. The time analysis was centered around two main phases, shown in Figure 1. I didn't worry much about the time for Phase 1: a previous system did that task on an IBM System/360 Model 25 in a minute, and the microprocessor on the personal computer was more powerful than that old workhorse. Instead, I concentrated on Phase 2, which I thought would be limited by the sixty-lines-per-minute speed of the printer. Each page of the report contained about thirty lines, so the total time of forty minutes was well within bounds. After this short analysis, the company purchased three personal computers and I implemented the design.

The first implementation of the program was revealing. Storing the BASIC program required about twenty kilobytes of main memory that I had ignored in my calculation; the safety factor of two saved the day. The forty minutes of printing time was right on the mark. Unfortunately, I was way off in the time to read the records and build the table. Instead of taking a minute, it took fourteen hours, which made it awfully hard to prepare a few reports a day. The problem was that I had compared assembly code on the old System/360 with interpreted BASIC on the personal computer, ignoring the fact that interpreted BASIC usually runs several hundred times slower than assembly code.

At that point I did a more careful back-of-the-envelope calculation. Using the parameters described above (1000 records of 80 columns each) and ballpark guesses at other parameters (50 BASIC instructions per column and one hundred BASIC instructions per second) gave the following:

(1000 Records) x (80 Columns/Record) x (50

Instructions/Column) / (100 Instructions/

Second) x (3600 Seconds/Hour) $\approx$ 11 Hours

Alternatively, I knew that the old machine took one minute for the task and executed an instruction in about ten microseconds. The slowdown to ten milliseconds is a factor of one thousand, and one thousand times the previous value of one minute is about seventeen hours.

Had I known the expense of this approach before I built the program, I would have used a faster language. Instead, I had an existing 600-line program and no choice but to tune the code. The 70 lines of code in Phase 1 accounted for over 90 percent of the run time, and just 3 lines accounted for 11

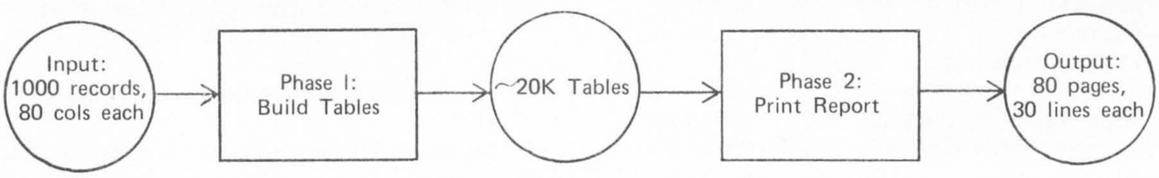

Input: 1000 records, 80 cols each
Phase I: Build Tables
~20K Tables
Phase 2: Print Report
Output: 80 pages, 30 lines each

Figure 1

hours (less than one percent of the code took 75 percent of the time!). I spent forty hours replacing 70 lines of BASIC with 110 lines of BASIC and 30 lines of assembly code; that reduced the time of Phase 1 from fourteen hours to two hours and twenty minutes. That was good enough for this particular system, but more than it might have been had I done a quick calculation beforehand and then chosen a more efficient implementation language.

Quick Calculations in Everyday Life

When you use back-of-the-envelope calculations, be sure to recall Einstein's famous advice.

"Everything should be made as simple as possible, but no simpler."

We know that simple calculations aren't too simple by including safety factors to compensate for our mistakes in estimating parameters and our ignorance of the problem at hand.

Douglas Hofstadter's "Metamagical Themas" column in the May 1982 "Scientific American" is subtitled "Number numbness, or why innumeracy may be just as dangerous as illiteracy"; it is reprinted with a postscript in his book "Metamagical Themas," published by Basic Books in 1985. It is a fine introduction to ballpark estimates and an eloquent statement of their importance.

Physicists are well aware of this topic. Jan Wolitzky has written:

> I've often heard "back-of-the-envelope" calculations referred to as "Fermi approximations," after the physicist. The story is that Enrico Fermi, Robert Oppenheimer, and the other Manhattan Project brass were behind a low blast wall awaiting the detonation of the first nuclear device from a few thousand yards away. Fermi was tearing up sheets of paper into little pieces, which he tossed into the air when he saw the flash. After the shock wave passed, he paced off the distance travelled by the paper shreds, performed a quick "back-of-the-envelope" calculation, and arrived at a figure for the explosive yield of the bomb, which was confirmed much later by expensive monitoring equipment.

One reader told of hearing an advertisement state that a salesperson had driven a new car 100,000 miles in one year, and then asking his son to examine the validity of the claim. Here's one quick answer: there are 2000 working hours per year (50 weeks times 40 hours per week), and a salesperson might average 50 miles per hour; that ignores time spent actually selling, but it does multiply to equal the claim. The statement is therefore at the outer limits of believability.

Everyday life presents us with many opportunities to hone our skills at quick calculations. For instance, how much money have you spent in the past year eating in restaurants? I was once horrified to hear a New Yorker quickly compute that he and his wife spend more money each month on taxicabs than they spend on rent. And for California readers (who may not know what a taxicab is), how long does it take to fill a swimming pool with a garden hose?

Gathering Lobsters

Several readers commented that quick calculations are appropriately taught at an early age. Roger Pinkham of the Stevens Institute of Technology wrote:

> I am a teacher and have tried to teach "back-of-the-envelope" calculations to anyone who would listen. I have been marvelously unsuccessful. It seems to require a doubting-Thomas turn of mind.
>
> My father beat it into me. I come from the coast of Maine, and as a small child I was privy to a conversation between my father and his friend Homer Potter. Homer maintained that two ladies from Connecticut were pulling 200 pounds of lobsters a day. My father said, "Let's see. If you pull a pot every fifteen minutes, and say you get three legal per pot, that's 12 an hour or about 100 per day. I don't believe it!"
>
> "Well it is true!" swore Homer. "You never believe anything!"
>
> Father wouldn't believe it, and that was that. Two weeks later Homer said, "You know those two ladies, Fred? They were only pulling 20 pounds a day."
>
> Gracious to a fault, father grunted, "Now that I believe."

Lifelong Inquisitiveness of Children

Several other readers discussed teaching this attitude to children, from the view-

(please turn to page 26)

Computer Programming in Nicaragua

Peter Torvik
47 Maple Ave.
Cambridge, MA 02139

"The Nicaraguan government agency concerned with nutrition uses a dBase III application to track malnutrition throughout the country."

Recently I taught a course in C language programming to employees of the Nicaraguan National Directorate of Informatics (DNI). DNI is the government agency responsible for setting computer policy for Nicaragua, with emphasis on the selection of appropriate hardware and software. The effects of war, underdevelopment and a difficult economy present severe obstacles to computer users in Nicaragua, but substantial efforts are being made to use computers to better conditions there.

Situation of Computing in Nicaragua in July 1979

When dictator Anastosia Somoza left the Central American nation of Nicaragua for exile in Miami, Florida in July, 1979, a turbulent and destructive decade came to an end. The capital city, Managua, was devastated by an earthquake in 1972 from which the country's infrastructure and economy had never recovered. 45,000 Nicaraguans had been killed and 160,000 wounded in the course of the struggle which led to Somoza's fall.

Throughout the 1960s, the United States supplied large amounts of economic and technical assistance to Nicaragua. The economy of the nation was oriented toward agricultural production on large farms for export to the United States. Computerization was quite primitive. Approximately 70 obsolete IBM minicomputers and mainframes were in the country. Burroughs had withdrawn from Nicaragua in 1977, due to the civil war, leaving three mainframes unsupported. IBM had continued to support systems, but imported no new equipment.

The most serious problem for computing in Nicaragua, however, was a human problem. Much of the middle and upper class had left Nicaragua, including many of the country's computer professionals. The civil war also took a heavy toll. Before 1979, malaria, polio, measles and other diseases were rampant, and average life expectancy was 51 years. This history is visible today in the streets and offices of Managua. Half the people are under fifteen years of age, and almost everyone is remarkably young for their position. My students were in their late teens and early twenties, with the youngest still working on his high school diploma. Many of the nation's leaders and managers are not much older.

Somoza once said, "I don't want educated people. I want oxen."

Education was not widely available in prerevolutionary Nicaragua, and in 1979, 52% of Nicaraguans were illiterate. After the exodus of much of the elite, a shortage of educated people impeded technical progress. An aggressive literacy campaign reduced the illiteracy rate to 12%, and a followup campaign is underway, but education remains a problem. It seems as though every one in Nicaragua is enrolled in night school, and the DNI staff were no exception. The lead programmer taking my course, for example, was pursuing a degree in economics at night. In fact, school enrollments at all levels have more than doubled since the revolution.

Volcanos and Storms

Smoldering volcanos, rather than skyscrapers, mark the skyline of Managua, and my first day of instruction was interrupted by a set of tremors. Thunderstorms mark every afternoon for the half of the year which is the rainy season.

Nicaraguan computer users are accustomed to a precarious power system. Surge protection is absolutely mandatory for any computer installation, and uninterruptible power supply units are extremely helpful. They add to the cost of computer installations and are in scarce supply, but their numbers are growing.

In spring 1987, the United States Central Intelligence Agency began openly providing explosives, instruction and maps to Honduran-based rebels, called "contras", for attacks on electric power installations. It was in an attack on a hydroelectric installation in northern Nicaragua that the engineer Benjamin Linder became the first US citizen killed while supplying technical assistance to Nicaragua. More than 8 Europeans (British, French, West German, Spanish, ...) assisting Nicaragua have previously been killed. Transmission towers in the Managua area have also been struck, and these attacks now add to the earthquakes and storms which impact productivity.

"There Isn't Any"

Familiar words to anyone who has lived in Nicaragua are "no hay" -- Spanish for "there isn't any." "No hay" applies to everything, from the dish you had set your heart on for dinner to paper for a printer. Many factors contribute to serious shortages of everything which cannot be produced within the country. The economy was ravaged during the 1970s by the earthquake and civil war that made Somoza leave. At the same time, economies throughout Latin America were being devastated by the rapid rise in oil prices and the collapse of the prices of the raw materials which those countries produce. Since the revolution, the United States government has also pursued policies designed to damage the Nicaraguan economy. Rebels, the "contras," have been organized and supplied by the US government to continue the old civil war, particularly destroying equipment, crops, and infrastracture, and diverting much of the national budget into defense.

In 1985, President Reagan prohibited US trade with Nicaragua, cutting off US markets to Nicaraguan products, and cutting off the United States as a source of spare parts. Because the United States had been a major trade partner, this was highly damaging to the Nicaraguan economy. The embargo affects everything from food to building materials, but a vivid illustration is glass bottles. Bottles are not made in Nicaragua, so Coca-cola is sold on the street in plastic bags. To buy rum in a store, you must bring your own bottle. In a hotel, we paid a deposit of about five dollars for a bottle.

Computer users have suffered heavily -- printer paper, ribbons, and floppy disks are in chronic short supply. In the university, professors often teach without textbooks, or a single manual may be shared by an entire class. Chalk for my blackboard was carefully rationed, and I would take care to use every last bit of it. Computer parts are expensive and slow to arrive: a print head for a Monroe printer at the National Bank which would cost $115 in the United States costs $500 in Nicaragua, and could take a month to get.

Role of Computers in Society

In a sense, the role of computers in Nicaragua is no different from anywhere. Payrolls, social security systems and banks are constant throughout the world. Computers can do work faster and cheaper; so Nicaraguan programmers automate these functions. Although North Americans typically think of Third World countries as overpopulated, Nicaragua has a severe labor shortage, and automation is a solution. Because economic problems dominate the national agenda, the various banks and ministries responsible for the economy struggle to apply computer technology to the management of the economy.

Many computer projects in Nicaragua, however, are different from comparable solutions in the developed countries. An interesting project which is being considered by a Nicaraguan bank illustrates the way technology is being applied in a uniquely local way.

Nicaragua's geography presents problems for economic development. Population is concentrated in a small, fertile strip along the Pacific coast. Much of the middle and north of the country are mountains, while the east, aptly named the Mosquito coast, is largely jungle. Historically, the eastern coast has been physically and culturally isolated, and economically underdeveloped. Only a single, hazardous road connects the two coasts, and the eastern city of Bluefields is only accessible by air or boat. Telephone service beyond the Managua area is still limited and quality is poor. Creating a unified banking system which serves most of the country poses serious problems. So, last summer a US advisor and his Nicaraguan counterparts were at work designing a packet-switching network which would use radio as the delivery mechanism.

The use of computers in Nicaragua is not confined to banking. The Ministry of Health uses a microcomputer to cope with shortages of medical supplies by tracking stocks of antibiotics and equipment throughout the country. The Nicaraguan News Agency, with offices in the United States and several Latin American countries, plans to use a PC, modem and electronic mail to provide an affordable version of a wire service to the Nicaraguan newspapers. The newspaper "Barricada" is evaluating computer systems to replace its antiquated typesetting equipment. PAN, the government agency concerned with nutrition, uses a dBase III application to track malnutrition throughout the country, and the agrarian reform agency uses a microcomputer to determine the effects of farm credit and loan policies on agricultural productivity.

Nicaragua's Strategy for Guiding and Controlling the Computer Revolution

Everyone with computer experience knows that technical projects can go awry. Sometimes projects founder, wreck budgets and consume the attention of technical people, managers and users for many times their planned span. Wrong specifications and designs produce expensive systems which are never used, or have to be completely redone. Training, documentation and future maintenance are often problematic. On a higher level, tragedies like those of Chernobyl and the Challenger illustrate what can happen when technical projects are out of control.

A wealthy nation like the United States can tolerate and dissemble these problems. Where many urgent needs compete for the attentions of only a few computer people and a few computers, projects that fail are very serious concerns. The shortages of technical talent, of access to technical assistance, and of sufficient educational programs to keep engineers abreast of the current developments in their field make the potential for trouble even greater.

On the bright side, there is a saying that "the pioneers are the ones with the arrows in their backs." The Nicaraguans are not first, and they learn and learn well from the experiences of the countries which preceded them into the computer age. There are certainly areas where the Nicaraguan computer revolution is proceeding on momentum -- where investments had already been made, and where equipment and systems are already in place. But every reasonable effort is made to use whatever works. New projects, however, are being guided by a sensible strategy.

Nicaragua is concentrating on defining and mastering a small set of standard tools and applying them to most problems. This way, technical people become productive more quickly, and the small number of skilled computer people available are able to apply their skills widely. The emphasis is on using ingenuity to solve a problem with the tools at hand, rather than on picking the absolutely "best", most efficient tool for every problem. At this point, dBase III+ and Lotus 1-2-3 are being used in Nicaragua to solve most problems. The COBOL programming language is also part of this skill set.

The C language and the Unix operating system are of interest to Nicaraguans, because they might fit in with this strategy at a later point in the process of computerization, where the packages and COBOL prove insufficient. They are being considered at DNI, and several North American computer people have provided instruction. C is being used on a few microcomputers, some Xenix systems have been installed and C and Unix are being taught in the universities.

Computer hardware is also being standardized. IBM PC-compatible microcomputers are favored for most jobs, because of their reliability, simplicity and availability. Last June there were 291 microcomputers of various sorts in Nicaragua (a few years ago there were only about 20 microcomputers in the country), and the number was growing at the rate of about 20 per month. Fujitsu-built minicomputers marketed by a Spanish company are also being imported, and one of my students had spent several months in Madrid at a computer school.

Educational Conditions

A few years ago, the University of Central America (UCA) in Managua had one hundred and eighty-nine students enrolled in the only degree program in computer science in Nicaragua. The university had a single IBM System/32 which was used to administer the entire university and for teaching computer science. Ninety percent of the students graduated without using a computer. Conditions have improved dramatically -- the National Autonomous University (UNAN) now offers the degree program in computer science, and both UCA and UNI (the engineering university) offer computer instruction. UNAN has enough PCs that students can be sure of spending some time each semester in front of a computer. The UCA System/32 has also been replaced by PCs, and UNI has a collection of about 20 assorted microcomputers.

My class had access to one Compaq PC for three hours most afternoons. A single Spanish copy of a C text, and two manuals in English were the only books. Only one student had sufficient command of English to use manuals, so the students had to take turns with a single textbook and PC. As some of the very few people in the country with technical training, the students were also juggling heavy work loads, and sometimes had to rush off to critical projects. Lectures must play a greater role in teaching from practice. In comparison to teaching from theory in the United States, it was critical to impart problem-solving skills and techniques through the lecture and to attempt to find shortcuts to the lessons that students usually learn for themselves in the course of programming.

The students were remarkably bright. The portion of the course material which concerned the C language itself moved quickly, and the students were writing working programs very quickly. More advanced programming problems, however, encountered serious conceptual barriers, as the students were unprepared to work comfortably with abstract concepts in such important areas as data structures. Assembly language experience is virtually unknown among Nicaraguan programmers; so the C language represented an extreme of abstraction. Followup courses are stressing these skills.

Some people cite these sorts of problems as proof that any efforts to do computer programming in a developing nation such as Nicaragua are inappropriate, and recommend that only standard commercial products such as spreadsheets and databases be used. The Nicaraguans, however, appear determined to proceed carefully, but with all due speed, to explore programming and to develop expertise in whatever they decide is useful and manageable.

The Center for Training in Informatics and Systems

Recently, DNI opened the Center for Training in Informatics and Systems (CAIS). This center exemplifies Nicaragua's computerization strategy, and addresses the educational problems. The center, when fully equipped, will contain fifteen Canadian-supplied IBM PC/XTs. It will be staffed by five trainers, and employees from the government and private sector will receive training in dBase III, Lotus 1-2-3, and word processing. Courses will be given at various levels, including courses for new trainers. Advanced students will receive instruction in COBOL, basic computer repair, and structured programming. Long term plans call for the creation of additional centers to specialize in repair and more advanced programming and system design skills, and for workshops in other cities. Organizers compare this to the literacy campaign, in which students travelled to remote parts of the country to teach peasants to read and write.

The CAIS center has opened, and a trainer from the US went to Nicaragua in June 1986 to complete the training of the center staff and deliver training materials supplied by a US training company. The $25,000 needed to complete the center is being raised in the US.

Foreign Technical Assistance

While the Nicaraguans have retained control of the decisions surrounding the computerization of their country, and concentrated their efforts on developing Nicaraguan technical expertise, they have received a great deal of outside assistance. TecNICA, a California-based organization, arranged my work in Nicaragua. TecNICA has provided more than 350 advisors in Nicaragua, and maintains a full-time staff in Managua. TecNICA organizations in the US, Canada and Mexico City also work on software development and technical translating projects and locate supplies and equipment. The bulk of TecNICA's assistance has been in the computer field. but their assistance has also included other technical fields.

Other US organizations also provide technical assistance in Nicaragua. Science for the People places computer science instructors in the Nicaraguan universities and NICAT (Nicaragua Appropriate Technology Project) places some US engineers, especially in areas like electrification. A Nicaraguan leader estimated that there are four or five thousand US citizens in Nicaragua at any given time -- about one thousand of them permanent residents of Nicaragua, 1200 on assignments ranging from six months to three years, and the balance on short term projects and tours. Hundreds of these people, organized as the Committee of US Citizens Living in Nicaragua, have demonstrated every Thursday morning since the 1983 invasion of Grenada outside of the United States Embassy in Managua, and publish a newsletter, "Through Our Eyes," for distribution in the United States.

Massive International Assistance

A number of other nations provide public or private assistance to the Nicaraguans. The dramatic improvements in computer science education at the Nicaraguan universities are largely due to the aid of private European groups. West German, French and Spanish groups donated the computer equipment at the engineering university (UNI), and French groups also donated the computers for the Autonomous University (UNAN). A West German group placed a professor at UNI for three years, and many European computer science teachers come to the universities as guest professors. The choice of Spain as the primary source of Nicaraguan minicomputers was a result of generous terms, and the government of Peru has recently offered technical aid, including IBM PC/XT-compatible microcomputers.

The massive international assistance to Nicaragua has not been limited to the computer field. Dozens of nations provide significant aid, including Mexico, Switzerland, West Germany and China. Churches and international relief organizations such as Oxfam supply aid. I met specialists in various fields from India, Canada, Great Britain, Denmark, and Spain. Individuals from Australia, Austria, West Germany, Switzerland, France, Belgium, Italy, Holland, Sweden and Argentina also work in Nicaragua as advisors. Cuba and the socialist countries supply some assistance in the computer field, but only a limited amount since the Nicaraguans have standardized on US and Japanese technology. The socialist countries supply more help in other areas -- for example, a TecNICA geologist specializing in volcanoes worked with Eastern European geologists and Soviet measuring equipment.

Deaths of Foreign Advisors

US hydroelectric engineer Benjamin Linder was not the first foreign advisor killed by the "contras". At least eight foreigners, all from Western Europe, have been killed, and many more kidnapped, while providing technical assistance in Nicaragua. On June 14, 1985, a West German forester Ms. Regine Schmemann was kidnapped with two Nicaraguan foresters and taken into Honduras. Due to international pressure, Schmemann was released; the Nicaraguans were not. Eight more West German hostages were taken in May of 1986 and held for 25 days. Doctor Pierre Grosjean of France was killed March 26, 1983. A Swiss agricultural expert, Maurice Demierre, was killed by a contra landmine on February 17, 1986. Ambrosio Mogorron, a Spanish health worker, was killed by a contra landmine on May 24, 1986. Belgian civil engineer Paul Dressers was machine-gunned on June 4, 1986. Yvan Leyvraz, a Swiss development expert, Bernhard Kalberstein, a German water project engineer, and Joel Fieux, a French communications technician, were killed in an attack on July 28, 1986.

Because of these deaths, the Nicaraguan government in August, 1986 removed foreign workers from the most dangerous parts of the country. Linder and other US advisors had applied for special permission to work in the zone where he was killed. Many advisors in Nicaragua, including myself, had believed that the contras would be careful to avoid killing a US citizen, for reasons of public relations. Even Elliot Abrams, the US State Department's chief apologist for the contras, had supported this view just a few months before Linder was killed, when he said "acts of terrorism are crazy. They're counterproductive. Not only are they immoral, they're stupid from the point of view of a guerilla army." (Unfortunately for Benjamin Linder, Abrams proved to be lying.) Foreigners, to be sure, have been only a tiny portion of the victims -- in 1985, for example, the contras killed, wounded or abducted 4,770 Nicaraguans.

The Position of Informatics Professionals in Nicaragua

A poster in a classroom of the Nicaraguan Central Bank reads: "To become a technician is not a credential for acquiring privileges, but a social responsibility."

The very poor have gained the most from the revolution through land reform, health care, education, and rising standards of living. One of my students described his country as a pyramid, with the very rich owners of the big cotton and coffee farms and factories at the top, and the poor at the bottom. In the middle lie the technicians, managers and engineers, making salaries of about 30 dollars a month. The three digit inflation rate and chronic shortages cause their salaries and standard of living to constantly erode. Their counterparts in the United States make a comfortable living, working with modern equipment and fascinating technical challenges. Nicaragua's economic problems are so severe that only half the workforce is still in the salaried sector -- more and more workers are leaving the formal economy to engage in speculation or informal trade. The Nicaraguan computer professionals, by their superior education, have more options than most -- including the option of

(please turn to page 26)

Accidents — Independent and Chained

Prof. John Boag, Emeritus
Dept. of Physics
London University
London, England

"Nuclear reactor safety engineers have to envisage as many as possible fault sequences as their imaginations can devise -- but of course there are limits to their imaginations."

Accidents That Happen Often

The only accidents whose probability we can assess with reasonable confidence are those that happen frequently. Among these are accidents in the home, on the road, or in industry. In these instances we have a large population of events from which we can deduce by reliable statistical methods the probability of an accident occurring under given circumstances. We can, in principle, carry out a cost-benefit analysis and decide, for instance, whether the convenience of driving is worth the risk of death or injury on the road. Few people do this analysis, but the data for it are available.

Accidents That Happen Rarely

When we consider rare accidents which could involve far greater damage to life and property, it is obviously desirable that we make some attempt to calculate the probability of such an event. But we cannot do so on the basis of their observed statistical frequency. There are too little data, and we must approach the calculation in a different way.

A system as complicated as a nuclear reactor is built from many components, some large and some small, such as electrically operated pumps, relays, and valves. For each of these separate components, one can determine from lengthy testing, or experience in practical use, a good estimate of its reliability. Components with special responsibility for maintaining safety can be duplicated or triplicated so that if one item fails, another immediately takes over. By such design precautions and a knowledge of the failure probability of the individual components, the designer will estimate a very low probability of accident for the whole installation.

Safety Engineer Estimates

It is the task of safety engineers to envisage as many possible modes of failure as their imaginations can devise and to see that automatic systems can respond adequately to all of them. Such calculations inevitably arrive at very optimistic reliability estimates for the whole system. If they did not, the engineers would introduce further safeguards until they could make an optimistic estimate. Thus, one can find, for example, ludicrous estimates of nuclear reactor reliability such as "one serious accident in one million years of operation."

I am not implying that these elaborate calculations are unnecessary. They are an essential part of the analysis that aims to ensure that the equipment is as safe as possible.

But events have shown that calculations ignore the most significant of all causes of accident -- human error. And how is one to assess the probability of accident from this source? Or even imagine all the foolish actions of which a human being is capable? What probability would one have assigned to the action of the Chernobyl operators in disconnecting all the safety systems before commencing their ill-fated experiment?

"Common Mode Failure"

There are other weaknesses in assessing the reliability of a system based on the reliability of its separate parts. If electrically operated pumps, relays or other components are backed up by duplicates, both could fail simultaneously if they have a common electrical supply and it fails. This is called "common mode failure," and there are other types of "common mode failures" that are not recognized and guarded against as easily as the failure of a common electrical supply.

In a high pressure reactor, for example, the integrity of the welded steel pressure vessel has to be taken for granted. This component is too large to be tested to destruction in order to gain information on its ultimate strength. The designer has to depend on calculations and on careful supervision of welding during manufacture. However, no pressure vessel could be made strong enough to withstand the pressures that could be generated in a runaway reactor like the one at Chernobyl. So the fracture of the pressure vessel, perhaps under conditions far more severe than the designers ever envisaged, can be regarded as a "common mode failure."

In principle, of course, a reactor could be made fully automatic, using only computers and relays to control its operation. That would still not preclude the possibility that a skillful operator could intentionally override the automatic systems, as happened at Three Mile Island and Chernobyl. It has generally been believed that the presence of an operator increases safety more than it increases risk, and there are numerous instances to support this view.

A Real Accident Sequence

For example, an accident sequence at an American reactor ran as follows: The reactor commenced a sudden "excursion," i.e., a spurt of power that could carry it beyond safe limits. The operator observed this and reacted by immediately pressing the "scram" button which shut down the reactor. The reactor was restarted, and some time later another "excursion" commenced. This time the operator did not immediately press the "scram" button. He left it to the automatic equipment to shut down the reactor. However, the excursion continued for some 25 seconds, and the thoroughly alarmed operator then scrammed the reactor manually.

An investigation revealed that the automatic scram had failed on the first, as well as on the second, occasion. A subsequent inspection of the equipment traced the problem to two conventional relays which had become so clogged with dirt and grease, due to lack of routine maintenance, that they did not open on the appropriate signal. This, too, can be ascribed to human error, for the nominal reliability figure for the relays assumes implicitly that they are regularly maintained and inspected. In this particular incident, a human operator did prevent an accident. Conversely, at Three Mile Island and Chernobyl, intervention by the operators was the principal immediate cause of the accident.

Authorities Contemplate Only Some Accidents

I have said that reactor safety engineers have to envisage as many possible fault sequences as their imaginations can devise. But, of course, there are limits to their imaginations. There used to be a good deal written about "maximum credible accidents." This designation, alas, begged the whole question. It merely indicated the maximum accident the safety engineers were prepared to contemplate. The events at Three Mile Island and Chernobyl would have fallen well outside the maximum credible accident scenario. There is a grave danger that in the nuclear weapons field the authorities are prepared to contemplate only those accidents that they reckon they can cope with.

The official report on the Three Mile Island accident (the Kemeny Report) identified what it called the "mindset" of practically all those involved (from the control room operators to the senior members of the nuclear safety inspectorate) as an important background reason for the accident. Long years of operation without an accident that fell outside the maximum credible accident limits had lulled them all into regarding the reactor as "tame," that is, easily controlled. Such an attitude, or mindset, blunts the imagination and encourages laxity in the observation of safety rules. This attitude has also been identified as an underlying cause for the total disregard of safety precautions which occurred at Chernobyl. The planned experiment was not even recognized by the plant management as involving any safety hazard.

Stereotype of a Ruthless Enemy

We can surely recognize an analogous mindset among those politicians and military leaders who credit nuclear weapons with keeping the peace in Europe for 40 years. In fact, what the nuclear arsenals have done is quite the opposite. They have maintained international tension during those 40 years, bringing it at times (like the Cuban missile crisis) very close to the boiling point. The development of new types of military hardware has been a principal reason why little or no agreement has been reached on limiting the build-up of nuclear arsenals.

The US administration's fixation on the Strategic Defense Initiative (SDI) is only the latest instance of this mindset, the false logic of which must be exposed whenever the argument is put forward. The possession of weapons by both alliances inevitably demands the creation of the stereotype of a ruthless enemy, bent on using them. The mindset that nuclear weapons are a contribution to keeping the peace must be opposed even more forcefully than the idea that one can safely play games with nuclear reactors.

Common Emotional Disturbance

I have spoken of "common mode failure" as a serious threat if it is overlooked in complex mechanical or electronic systems. An analogous situation could exist at the human level, among the personnel of a military installation. Work in the confined space of a nuclear submarine, for example, could produce a highly dangerous psychological mindset in which the custodians of nuclear weapons might, at a time of high international tension, suffer a common emotional disturbance. Action without orders or contrary to orders would be possible.

Drug abuse is also a latent danger and another possible "common mode failure." So is religious fundamentalism -- the "Armageddon Complex." If communication channels broke down at a time of heightened alertness, or if instructions were garbled and misunderstood, the captain and his officers might ultimately assume responsibility for firing their missiles. This scenario, of course, is dismissed as incredible by military authorities, just as catastrophic accidents to reactors exceeding the maximum credible level were dismissed by civil nuclear reactor operators. But as we have seen, such accidents can occur.

There have already been many serious accidents involving nuclear weapons and their delivery systems. Several weeks ago, a Soviet nuclear submarine sank to the bottom of the Atlantic after an internal explosion. We have since learned that this was not the first such event. Official US publications list accidents involving American nuclear weapons carriers in the period of 1956 to 1986. Most of these incidents refer to bombs accidentally dropped from airplanes or lost in plane crashes. The Palomares incident in 1966 and the Thule incident in 1968 are two of the most serious, and both caused radioactive contamination. I have no comparable data for the Soviet nuclear forces, but it is reasonable to suspect that similar incidents have occurred in the USSR.

The Risk From Software

I have addressed the danger of equipment failure, the hardware risk, and the ever-present risk of human error. What about the risk from the software -- the computer programs, often of great complexity, that are built into every sort of modern military equipment? Computer programming has grown up as an art rather than a science, with individual experts often using short cuts of their own design. "Debugging," the process of detection and elimination of accidental errors or logical confusion, is an integral step in software development. But it is not easy to find all the bugs.

If SDI were ever to reach the deployment stage, which seems unlikely, the computers would require programs of such inordinate length and complexity that it would be impossible to check them for overall accuracy. And live tests on the system would obviously be impossible since this would involve firing nuclear explosions in space. Yet, to be effective, the system would have to operate with nearly 100 percent efficiency the first time it was ever activated by attacking missiles, perhaps after years or decades of non-use and, perhaps, inadequate maintenance.

The mind boggles at the total impracticality of the concept. It is surely a sad instance of human error that fixation upon this mythical umbrella prevented agreement in Reykjavik on the most comprehensive arms reduction proposals yet put forward.

So what lessons for the prevention of nuclear war can we learn from nuclear reactor accidents? The chief lesson is that, no matter how elaborate the precautions, something will, in due course, go wrong. And when it does, no limits can be set for the amplifying effect of human error or of deliberate contravention of safety rules or military instructions. In the case of civil nuclear reactors, the consequences, as we have seen, are limited in extent and duration. Many people may suffer, but civilization is not destroyed. A cost-benefit analysis may indicate that the energy we need can be obtained at lower cost by other means. Or it may not, for other ways of capturing energy also incur their peculiar risks.

Accidents in Weapons Installations

The type of equipment, the automatic safety devices and computer controls, the operating rules, and the kind of technical personnel involved in the nuclear weapons field are not very different from those found in

(please turn to page 26)

Software Development Systems

Peter Freeman
Professor of Computer Science
University of California
Irvine, CA 92717

"If we understand the realities and concepts of our world, we are in a position to formulate some precepts, or rules of actions, that impose a certain standard of action."

Software Development

As with systems work, it is important to understand the ground rules under which a book has been written. This prologue addresses four topics that will help you profit from your reading: the general organization of the book, terminology, my assumptions about your background, and the context of my background.

Several years ago I set out to write a textbook that would tell you everything you needed to know about software development. What I soon discovered, which should have been obvious in the first place, is that what we know about software systems development is evolving much too fast to permit it to be frozen into a classic textbook that will remain largely unchanged for years. That is not to say that one can't write a good textbook at this stage (there are several), only it will age quickly.

In this book, then, I have tried to capture three things that will help you understand not only what you observe and experience, but will encounter in other books and papers: realities, concepts, and precepts. These are not simply listed, but rather are presented in the context of an underlying view that systems work is a system itself.

It is important to be realistic, to understand what really is, to sort out fantasies and wishes from what is factual. That is often difficult in the present world of software, and is especially difficult if you are a newcomer to the field. I present those realities that I have seen demonstrated enough to be sure of, for example, that carefully designed systems are easier to maintain and modify than those that are not. The ultimate reality is captured in the title, however -- that it is possible to make sense of software systems development, if you adopt a systems viewpoint.

Formulating Precepts

Concepts (or ideas or abstractions), if drawn from specifics, are essential to formulating strategies, organizing technical work, making decisions -- in short, doing the work of and managing the technical process. While managers typically have a good set of concepts to guide their work and technical people a good set to guide theirs, one of the primary gaps of knowledge in the software field is created by each group not understanding the others' concepts. My focus here is on bridging that gap.

If we understand the realities and concepts of our world, we are in a position to formulate some precepts, or rules of actions, that impose a certain standard of action. Where possible, I have tried to provide you with those precepts that I have seen followed successfully. For example, a highly successful precept in many organizations is to always apply some explicit inspection process to every software workproduct (specification, design, program, and so on) before declaring it finished.

In tune with the underlying message, the book is organized around a description and explanation of the parts of a development system. Where possible, I have provided some analysis of why things seem to work the way they do, given guidance or perspective on strategies and approaches that seem to work (or not), and commented on what I have observed being done (or not done) in many development situations.

The book is organized in a straightforward way: Chapter 1 explores in a way useful to neophytes and experts alike the question "What is software?"

The Meaning of "System"

Let's start with a term that permeates this book -- "system." Although one can provide precise, technical definitions, the meaning I employ here is the intuitive one: a collection of things related in a way that forms a coherent whole. In dealing with systems, there are always two questions: What are the elements? and What are the relationships between them?

"Software" will be used largely in a generic sense to refer to collections of programs, individual programs, designs or specifications for programs -- in short, any of the information that directly relates to the instructions and control data (but not the application data) that is loaded into the hardware to produce desired outputs. I will use "program," "design," "specification," and so on when I mean those specific items of software; the meaning should usually be clear from the context.

Without going off into a long philosophical treatise, let me note that this is a very broad definition. Certainly for some purposes (for example, defining languages for software development) one must be more precise, but here I want to focus on the overall process of reaching development objectives. In that context, it is important to include all the information and workproducts that bear on reaching our objectives.

Another nondistinction I will employ in many cases is to equate "system" and "software." There are some obvious exceptions to this equation: On the one hand a single, 50-line program is not a system; I think your common sense will guide you to be able to apply what I may be saying about systems to your single program situation (you will find that a surprising amount of what is true for systems is true for individual programs).

On the other hand, there are certainly systems that are not software; indeed, one of the tenets of system development is that to the extent possible you should proceed with the early stages of design without worrying about whether it will be realized in hardware or software. As with scaling down to small situations, common sense should guide you in scaling up to systems that include hardware and other nonsoftware elements.

Software Is the Critical or Dominant Element

There is another reason for closely identifying software and system, however. In many situations today, the systems we are developing are certainly "software-intensive" in that software is the critical or dominant element. Further, because of its complexity, if you can deal effectively with the design of a software system, then you can probably also deal with the design of a larger system containing other elements. Indeed, some of the techniques for dealing with software development are now being used in the development of the overall system or systems that may be realized in hardware (for example, VLSI design shares a lot with the design of a pure software system). Software development is an instance of systems development. Much of what we know about one applies to the other.

Finally, it is important to recognize explicitly (and keep separated) that I am using the word "system" in two very distinct, but related ways. I will often refer to the system being developed -- the software -- or even just to "the system." My reference is to the product being produced.

On the other hand, the underlying conceptual framework and key to controlling development revolves around the idea of "software development system," a collection of objectives, policies, people, techniques, tools, and information that form a system for producing systems. I may sometimes abbreviate "software development system" to SDS to reduce confusion (and save space), but I have resisted the temptation to create a new term to stand for SDS since I don't want to lose the close identification with the inherent understanding we all have of what a "system" is.

Who Are You?

Even when writing a book intended to serve the needs of a broad class of people, as I have here, an author must make some assumptions about the potential audience. In general, I have assumed that you are interested in understanding or improving the process of creating software, very likely because you are interested in improving the quality of the software produced. There are at least four categories of people I hope will read this: technical professionals, managers directly concerned with software development, managers concerned with other aspects of organizational activity but who must interact with the development activity in some way, and others who are simply interested in what is going on in an active and challenging segment of our culture. There are others, such as students, teachers, and consultants, who can identify closely with one or more of these categories as well.

The senior technical person will gain some insight into what his or her management does (or doesn't do) and thus be able to work with that part of the team better. The junior technical person can gather some idea of why the world is organized the way it is, even if sometimes he or she has little opportunity to change it.

The technical project manager will find many points of interest here. Of special interest to this person will be the principles relating to the use of technical methods and tools. All technical people will gain deeper insight into the nature of software and the processes used to produce it.

The General Manager

The general manager (a person perhaps two or three levels above the individual project manager) that is responsible for development activities may find the principles presented here as useful as anyone. Indeed, it is this person (responsible for organizing the overall process, allocating resources, and seeing that it all continues to run) who often has the greatest opportunity for utilizing the ideas discussed here. This person in many cases (if not most) has "come up the ladder" and thus has also served as a junior technical person, technical leader, project manager, and so on.

There is another type of general manager emerging in many organizations today -- the person who has a technical background, has come up through the ranks, but is not conversant with software and its problems. Typical is the executive in an engineering or technology-based company where, until recently, software was something that the people in the DP center dealt with. The rapid growth of importance of enabling technologies such as computer-aided design and manufacturing (CAD/CAM) and computer-integrated manufacturing (CIM) is forcing issues of software development strategies to the top of such organizations; in addition, many technologies and processes that once were based on some other technology (electronics or fluids, in the control area, for example) are rapidly becoming computer-based, which again means heavy involvement in software.

Utilization of this book is not limited to those concerned directly with managing or performing the development activity. Anyone (executives, board members, or simply the intelligent layman) who has an open mind and a desire to understand one of the most challenging intellectual tasks in today's world will find something of interest here.

No deep understanding of computers, electronics, mathematics, or programming is necessary. If you have some or all of this background, then you will interpret things in that context and perhaps be able to make connections that others will not. An ability to think logically and to keep separate means and ends, to be able to identify parts and connections, is required.

Likewise, no deep understanding of management or organizations is necessary. If you do have a stronger background in this area, then, as with the technical person, you may be able to take away a deeper set of insights. An ability to understand that management is necessary and even productive in every situation is required.

Who Am I?

I don't have all the answers. No one does. But I do have a set of experiences and a background that helps provide me with a certain perspective.

My starting point is the technical side. I have designed and built complex software (scientific applications, operating systems, research programs in artificial intelligence). My graduate training is in computer science and I am still an active researcher attempting to advance our understanding and capabilities in software engineering. As a professor of computer science and lecturer to professionals I try to educate students in basics (such as computer architecture) and their applications (for example, design methods for complex software systems).

I have been able to gain some insight into the application of software technology through extensive lecturing and consulting in industry, in addition to my direct experiences. This has brought me into close and continuing contact with those trying to use the new technology (and the old) to build software ranging from small one-person, two-week projects to those that can only be measured in kiloyears of effort. Through this contact, my own use or participation in the use of techniques, and assistance to those most vitally concerned with improving their software development systems, I have developed a set of observations of what works and what doesn't work.

We Need Not Repeat Mistakes

I have been in the software field continuously since 1961. I have seen some significant improvements in software technology (along with the obvious improvements in hardware technology) but am increasingly frustrated by their slow adoption. It is not just that they are research ideas which must be translated into usable form; I am speaking of those techniques that are in use today (and have been for a number of years in some cases) by some organizations but that are still ignored or rejected by others, not only to their economic peril but to the safety and convenience of the public at large.

We don't have to wait for breakthroughs. There is much available today. Nowhere is it written that we must all repeat the mistakes of our predecessors. Yet that seems to be what many are trying to do.

Much has been written about the challenges of information technology to mankind in general and to the relative position of national economies in particular.

There is indeed a new wave of technology coming, but we must learn to perform, manage, and utilize software development if we are to understand and control whatever comes next -- because it clearly will be software-like in nature and thus subject to many of the same problems and solutions.

Software Is More Than Programs

Five or ten years ago, intelligent people were not ashamed to ask "What is software?" Today, with stories about it appearing regularly on the covers of newsmagazines or on the front pages of daily newspapers, most people hesitate to ask what they fear is a stupid question.

It isn't and they shouldn't!

Indeed, many of the people that are daily concerned with software should be asking this question. All too often, they do not have a sufficient understanding of what software is to permit them to deal with it in an effective manner.

The problem is not that people don't know that software is just another name for computer programs -- most third graders as well as older people now know that. The recognition that people often lack is that software is more than just programs and that there is one overriding characteristic of it that MUST be attended to -- the fact that it is a system.

Before looking more deeply at the process of creating software, it will be helpful to explore some of the aspects of software itself. As the old adage puts it, "To master your enemy, you must know him!" Most of us can profit by enquiring more deeply about the nature of software.

What We Want Computers to Do for Us

Isn't software just programs? Only if you are a complete literalist, absolutely not believing in abstractions, programming on your own personal computer, and never coming back to programs you wrote last year, is this true. The question is not simply what software is, any more than the question is simply what the trade policy of a nation is. Rather it is how you think about it and what role it plays in a larger context that are the important issues.

From the perspective of a computer, yes, of course, software is a set of programs that tell it what to do. But the perspective of a computer is rather limited and has little to do, directly, with what we want computers to do for us. That is why thinking about software just as programs leads to so many problems, both in their creation and use.

Perhaps the most common misconception in the software field is that productivity is measured by how many lines of code are produced in unit time. The common joke in many software development shops is some form of "Quit trying to figure out what you are doing and write some code!" While there may be times (few, in my estimation) when such an order contains some degree of truth, such comments, invariably based on an underlying and strongly held belief, belie a view that it is only programs, in the narrow sense of executable programs, that can be used to measure progress or that represent productivity output on the part of a developmental staff.

What this leads to quite often is an attitude on the part of management and technical people alike that the only important thing is the production of code. This focus, clearly understood in psychological and pragmatic terms, conditions the entire environment to focus on the production of code.

The result, time and again, is the production of mountains of code that cannot be integrated to work as a system, or the building of a system which does not satisfy the needs of the customer even though it may work well technically.

What, then, is software if it is something more than just programs?

Software is:

- The brain and soul of a computer, not just a coat of paint
- The embodiment of the functions of a system
- The captured knowledge about an application area
- The collection of all the programs and data that are necessary to make a computer a special-purpose machine designed for a particular application
- All of the information (documentation) produced during the development of a software-intensive system

All of these characterizations contain some important elements of truth that will help you better understand the world of software even if you are already deeply involved (mired?) in it. Let's look briefly at each in turn.

For years, the belief in many organizations, especially those that manufactured computer hardware or other sophisticated electronic equipment that included computers (for example, military systems), was that software was something that was added on after the system -- the hardware -- was built; in short, it was viewed as a coat of paint that was put on the hardware after the real system design had been done. As people keep rediscovering, general-purpose computer hardware does nothing without software.

The Driving of the Development Process

An often-heard debate is whether systems design should be driven by the hardware constraints or the software requirements. Worse, it sometimes is not even a topic for discussion, with the result that an inappropriate choice is made.

Usually, but not always, the hardware characteristics are permitted to drive the overall design. This can lead to severe problems later when features of the hardware that are inappropriate to the system constraints must be compensated for in the design of the software (if possible). By analogy, this would be like choosing a French restaurant because it is close, irrespective of its menu or type of service, and then ordering a quick-service Chinese take-out. If it can be made to happen, it won't be easy (or cheap)!

I witnessed a slow, agonizing example of this in an organization rebuilding a large application system. The decision was made (perhaps implicitly) to rebuild it on the same type of hardware. As the design of the rebuilt application system proceeded, software decisions about the database management system were then constrained by the hardware choice. This, in turn, resulted in such a great expansion in the requirements for disk storage that ultimately (after lots of new hardware had been obtained) the entire project had to be scrapped because the expanded configuration could not physically fit in to the operations facility. A careful system design in the first place could have identified or even sidestepped some of the problems by calculating the resources needed to implement the system under alternative assumptions about the hardware.

Two realities of software development are illustrated here. First, hardware considerations all too often drive the development process out of a mistaken belief that they are the dominant cost and most inflexible design element; while they may be, one cannot really say without looking at the overall system. Second, the "debate" as to whether a particular effort should be hardware- or software-driven is a red herring; neither is the right approach because only with a systems approach that takes into account and evaluates both elements appropriately can a system be produced that meets the functional and operational requirements of the customer.

Software provides the active portion of the system, the part that makes the system seem "alive"; hence, the characterization of it as the brains and soul (or heart) of a system. While it is obvious that the underlying hardware is an essential ingredient of any computer system, the nature of general-purpose computer systems is precisely that they can do anything (and, thus, nothing) that a particular set of software instructs them to do.

Focus on the Applications of the System

The second characterization above captures a viewpoint that is crucial to creating a successful software system: the focus on the functions of an application system, not on the functions of the underlying hardware. After all, carrying out a set of application functions is ultimately the purpose of any computer system. Taking this viewpoint helps to direct attention to the instrumental objective of building any computer system -- to achieve some goals in a wider sphere (such as reducing inventory costs, controlling a machine, or creating a new service to sell).

Leonard Sayles and Margaret Chandler in their perceptive study of the NASA program that landed the first man on the moon point out how important the overall objective set by President Kennedy was to the success of the program. It was an objective that often served as a forcing function to get things done in order that the objective could be met. They also point out, however, that while having a clear main objective is important, it does not help very much in deciding what to do next on a detailed level.

The last two characterizations start to provide that detailed guidance. One of the reasons that it is unproductive to think of software as only consisting of programs is that, at least implicitly, this usually connotes only the procedural part of programs, leaving out the all-important data portions. For this reason, it is sometimes useful to focus on the fact that the software that makes up the application functions is composed of both data and functions.

There is a fine line here: Is a large database, that is, the data in it, not the programs that access it and process the data, software? Generally, the database itself is viewed as being outside the realm of software (but certainly not outside the realm of computer science, nor of systematic ways of building the collection of data) while the data that define a database and its accessors is a part of the software. This carries with it a connotation of software being an active and general-purpose element that is not tied to a specific set of facts (as would be the case with the data stored in an actual database).

Information Produced During Development

I once saw a good example of ignoring the data portion of a system -- and the development problems that can result. A group was developing a complex system that included tables that would be used to define the characteristics of an individual installation of the system. The design team was not familiar with the application area and felt that the design of the tables would be simple and belonged in the province of the application specialists since it was just data; consequently, little attention was paid to data during most of the design phase.

As the early versions of the system were brought up for testing, it was discovered that the tables were not only unexpectedly large and complex, but that the efficiency of the system depended critically on their design. A serious delay in the development of the system resulted while the data portions of the system were belatedly designed. The failure to understand the structure of this data component of the system and treat it equally with the process portions nearly resulted in disaster.

When focusing on the development process, all the information that is produced during development is of interest. The success of an effort to build a software-intensive system is thus directly tied to how well one deals with all of the information present, much of which is something other than directly executable programs or their imbedded data. Hence I am led to the conceptual generalization that software is all the information produced during development. Software is many things, but all are simply aspects of information.

The Importance of Maintenance

It may come down to a matter of semantics: some would like to reserve the name "software" to refer just to programs (and perhaps data); if so, then there is a lot of other information that is very central to what software developers do that must be accounted for and named in some fashion. If we include all the information relevant to a piece of executable software in the generic definition, then we will have to deal with that information in the same rigorous and systematic way that we must deal with executable software. This is essential to successful development, since if it is not done information is lost or altered, introducing errors.

A common example is the widespread tendency to not maintain software designs. A system is built from some sort of design document, but as modifications are made to the code, these are not recorded in the design. Later, if major changes are contemplated, they can only be made intelligently by considering the design -- which is unavailable because it has not been treated in a rigorous and systematic way.

It is just as counterproductive to include every scrap of information in the definition of software as it is to restrict it to only executable code. At a gross level, we can differentiate among several important classes of information that are involved in producing software: software representations (representations for short), software engineering knowledge (development information), and domain-specific knowledge (application information). Because it is important to have a well-grounded understanding of these classes, let's take a deeper look at them before proceeding. (While the following discussion may seem technical, it is really just a logical description and will provide you some important characterizations.)

Software representations include programs, detailed designs written in a program description language, architectural designs represented as structure charts, specifications written in a formal language, system requirements expressed in a combination of notations, or any one of hundreds of other possibilities. In short, any information that in some direct way represents an eventual set of programs and their associated data may be included in a software representation.

It would, of course, be simpler if we had different terms for those objects that are executable and those that merely describe executable objects. But we don't. Reserving "software" to describe only executable objects -- and having no other widely accepted term available -- puts us back in the situation of focusing too much on the executable objects in their final form and not enough on the earlier descriptions. Hence the breadth of my definition.

The second class, software engineering knowledge, is all of the information that relates to development in general (for example, how to use a specific design method) or information that relates to a specific development (for example, the testing schedule on a project). The information of this type includes project-related information, software technology information (methods, concepts, techniques), knowledge about similar systems, and the detailed information relating to the identification and solution of technical problems on the system being developed. More general information (for example, the notice of this month's professional society meeting or the next office party) is in the larger set that we are not concerned with in any explicit way.

Understanding the Application Domain

The third class of information that is essential to creating software is the domain-specific knowledge about the application area. The examples are everywhere: understanding of a physical process to be controlled, accounting rules, procedures for updating and changing employee records, and so on. While this information is clearly essential to creating software, discovering it and putting it into a useful form is usually the province of an applications specialist such as a process control engineer or an accountant or a business systems analyst.

As we have applied computers to ever more complex situations, however, it has become evident that our understanding of the application domain is quite often the biggest stumbling block to creating effective software; at a minimum, communicating the understanding of the applications specialist to the computer specialist has increasingly been identified as a key problem. Progress has been slow at improving this interface, as we will explore in more detail below.

Some "solutions" provide tools to the applications specialist so that he can directly build the software. Thus the motivation for many of the "fourth-generation" languages, prototyping schemes, and program generators. While they are very powerful for some situations and certainly permit us to break down the communication barrier between applications specialist and computer specialist in some cases, they do not solve all problems.

Specifically, there are situations in which there is simply too much to be known in each domain (the application domain or the computer domain) to permit one person to master both; a specialist is required in each, with good communication paths between them. That is the thrust of some of the modeling techniques such as structured analysis.

Let's consider again the three classes of information defined above, and look at their overlap. The overlap is generally small. For example, although we have quite a bit of knowledge about how to develop software (software engineering knowledge), such as principles for organizing programs, most languages (software representations) directly incorporate very little of this software en-

(please turn to page 26)

Bentley – *Continued from page 10*

points of both parent and child. Popular questions were of the form "How long would it take you to walk to Washington, D.C.?" and "How many leaves did we rake this year?" Administered properly, such questions seem to encourage a life-long inquisitiveness in children, at the cost of bugging the heck out of the poor kids at the time.

Based on Chapter 6, *The Back of the Envelope*, in *Programming Pearls* by Jon Bentley, copyright 1986 by Bell Laboratories, Inc., Murray Hill, NJ. The book is published by Addison-Wesley Publishing Co., Reading, MA; this excerpt is reprinted with permission.

Ω

Torvik – *Continued from page 15*

moving to another country where they could use their skills more profitably. Their stake in the revolution is purely altruistic. It is a moving testament to their professionalism and love for their country that they are so many, and they continue to struggle to use their skills to solve Nicaragua's problems.

Ω

Freeman – *Continued from page 25*

gineering knowledge. That is one reason for the interest in and high hopes for the new programming language Ada which was designed specifically to incorporate software engineering knowledge in certain areas.

The intersection between software representations and domain-specific knowledge is also small. A language such as FORTRAN obviously embodies some very general information about the domain of scientific calculations; COBOL embodies some information about business data processing; Ada was intended to contain some information about the domain of embedded systems. The problem is that the amount of information is very small in order to permit the languages to be applied to the largest possible range of applications.

Base *are Perspectives,*
publi on-Wesley Publish-
ing mission.

Boag – *Continued from page 18*

civil reactor installations. If accidents cannot be prevented in reactors, they can, and will, surely occur in weapons installations. But the consequences could be immeasurably greater. If the accidental release of a nuclear missile at a time of high international tension were to trigger a nuclear war, there would be no subsequent opportunity for a cost-benefit analysis. The cost could not be measured, and the benefits are none.

No Fingers on Nuclear Buttons

The lesson I draw from all this is that there must be no fingers on the nuclear button: that there must be no buttons which could initiate a nuclear holocaust. For no one can be considered wise enough or strong enough to withstand the pressures that would fall upon them in this ultimate crisis -- least of all, those politicians who insist on keeping nuclear weapons as an ultimate threat. These weapons and all other means of mass destruction must be relegated to the scrap heap as a totally unsuitable means of solving the problems that will continue to arise between the different parts of this one world. They must be replaced by the only methods that can be effective in this world of high technology -- dialogue, diplomacy, constructive compromise and, especially, cooperation to launch a joint attack on poverty, hunger and disease, wherever they they still exist.

Based on a report in *IPPNW Report*, April, 1987, of the talk by Prof. John Boag at the Second Regional European Symposium in Madrid, Spain. Copyright 1987 by International Physicians for Prevention of Nuclear War. Reprinted with permission.

Ω

Newsletter – *Continued from page 27*

Prof. Chellamuthu said the system evolved in the centre had solved many interesting problems regarding phonetic variations and linear script patterns of many Indian languages. A text in Hindi, Tamil, Malayalam, English or Bengali is fed into the computer and can be converted to any one of the five languages desired. Thus a Tamil novel or technical book can be much more easily translated into the other four languages.

The university plans to include Kannada, Telugu and Marathi scripts to its list.

Ω

Newsletter – *Continued from page 3*

● Large mirrors, needed to direct laser beams toward their targets, are "particularly vulnerable to radiation from other lasers" and might even be damaged by natural particles orbiting alongside them in outer space.

● Just to maintain a space-based SDI system -- to control altitude, cool mirrors, receive and transmit data, operate radars and so forth -- will require 100 to 700 kilowatts of continuous power. This in turn may require 100 or more nuclear reactors in space. The entire task requires "solving many challenging engineering problems not yet explored." If the SDI system ever had to be fired in a nuclear war, one-billion watts of power would be needed.

All these points have been made in the past by various critics of the SDI program. However, never have they been made in such fastidious detail or by so authoritative a group.

The panel was chaired by Nico Bloemgergen of Harvard University and Dr. C.K.N. Patel of AT&T Bell Laboratories. Members included scientists from the Air Force Weapons Laboratory; Sandia, Lawrence Berkeley, and MIT Lincoln Laboratories; the US Military Academy; Xerox, KMS Fusion, Inc.: Cornell, Stanford, Columbia, Caltech and the Universities of Washington, Illinois and Arizona.

AUTOMATIC CONVERSION OF THE SCRIPTS OF ENGLISH, TAMIL, MALAYALAM, AND BENGALI LANGUAGES

"The Hindu"
Mount Road
Madras, India
January, 1987

The Computer Centre of the Tamil University, under its project to evolve an integrated script conversion system, has made a "breakthrough," adding Hindi to its list of convertible scripts. Prof. S. Agasthialingam, Vice-Chancellor of the university, said the centre had already perfected the technique for automatic conversion of the scripts of the English, Tamil, Malayalam and Bengali languages.

Prof. K.C. Chellamuthu, Head of the Computer Centre, said the integrated script conversion system study was launched four years ago to examine the script patterns, interrelationships and variation in forms and the phonetic levels of different languages, using the computer. The centre aimed at devising a system of designing and developing keyboards in Indian languages for computers, teleprinters and other telecommunication equipment. The fifth generation computer project envisaged natural language interface as the primary media of communication and in the context of a multi-lingual country like India, this had tremendous potential for quick and easy translation.

(please turn to page 26)

CACBOL – *Continued from page 3*

these one-word utterances is a substantial challenge to artificial intelligence.)

20 CHALLENGES TO ARTIFICIAL INTELLIGENCE: MEANINGS OF RUN (List 870703)

Meaning	Example
1. finish in a contest	the horse ran second
2. pass freely	the rope runs in a pulley
3. unravel	the stitches have run
4. melt and flow	the solder has run
5. operate	the motor is running
6. elapse	the time has run
7. become	the well ran dry
8. total	the bill ran to $60
9. proceed	the story runs to 9 pages
10. be performed	the play ran for 8 days
11. get past	his ship ran the blockade
12. travel	the bus runs hourly
13. pass quickly	he ran his eyes over the page
14. expose oneself	he ran the risk
15. cause to flow	he ran the hot water
16. search out	he ran down the facts
17. collide	he ran into the bus
18. depart	he ran off
19. terminate	the text ran out
20. become used up	the cook ran out of butter

(Source: Neil Macdonald's notes) Ω

Opportunities for Information Systems

– Instalment 10

THE TRAINING OF HUMAN INTELLIGENCE

Edmund C. Berkeley, Editor

How are we to increase the intelligence of humans by a substantial degree? How can we make use of computer systems for this purpose?

Intelligence is defined in the dictionary as "capacity for reasoning, understanding, and similar forms of mental activity; sound thought; good judgement." Forms of mental activity include observation, study, experimentation, memory, computation, deduction, languages, and more besides. Many, but not all, of these mental activities are partially taught in schools.

Nowadays it is clear that if we want to increase some capacity of humans, the chief sensible activities are encouraging, motivating, educating, training, rewarding, obeying the law, and telling the truth. Take for example the new plague of the 1980s, AIDS, "acquired immune deficiency syndrome." This is a virus which kills people, which is spreading widely, and for which no cure is yet known in spite of intense current investigation. AIDS is an excellent example of a new threat to the human species, for which old behavior of many prior centuries will not work. The problem of dealing with AIDS will certainly train human intelligence.

A capacity to behave in a new way requires instruction and training. The behavior has to be taught, either by oneself or by some subdivision of social organization, such as a doctor, colleague, computer, or environment. The environment of a sea beach rather quickly trains a new diver how to snorkel without filling his nose and mouth with sea water.

Probably more than 1000 properties of human intelligence can be trained by computer systems, ranging over categories of identification, memory, deduction, behavior, planning, situations, and many others. But the time to do this kind of training in increasing human intelligence does not come free. It has to come from doing less of something else, such as reduced watching of television or sports.

The market for computer systems that train human intelligence should be huge.

Ω

Games and Puzzles for Nimble Minds and Computers

Neil Macdonald
Assistant Editor

NUMBLE

A "numble" is an arithmetical problem in which: digits have been replaced by capital letters; and there are two messages, one which can be read right away, and a second one in the digit cipher. The problem is to solve for the digits. Each capital letter in the arithmetical problem stands for just one digit 0 to 9. A digit may be represented by more than one letter. The second message, expressed in numerical digits, is to be translated using the same key, and possibly puns or other simple tricks.

NUMBLE 8707

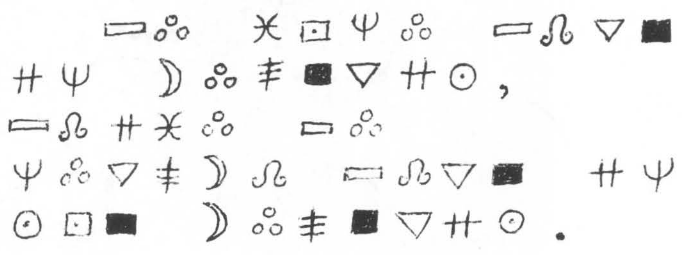
```
          N O T
     *    A N Y
     ----------
        T E T T
      L I Y A
    L N S O
  -------------
  = Y E T O O T
```

22730 91093 85273

MAXIMDIDGE

In this kind of puzzle, a maxim (common saying, proverb, some good advice, etc.) using 14 or fewer different letters is enciphered (using a simple substitution cipher) into the 10 decimal digits or equivalent signs, plus a few more signs. The spaces between words are kept. Puns or other simple tricks (like KS for X) may be used.

MAXIMDIDGE 8707



SOLUTIONS

MAXIMDIDGE 8705: The end of mirth is the start of sadness.

NUMBLE 8705: Even the King's men sin.

Ω