computers and people

formerly *Computers and Automation*

Sept.—Oct., 1987

Vol. 36, Nos. 9-10

COMPUTER GIVES BODY TO 6TH CENTURY REMAINS

The Computer Almanac and Computer Book of Lists — Instalment 55

Neil Macdonald, Assistant Editor

10 QUESTIONS FOR CHECKING YOUR ORGANIZATION'S EXPOSURE TO COMPUTER FRAUD AND COMPUTER CRIME (List 870901)

Initiating Transactions:
Are data processing employees prohibited from initiating original accounting transactions, adjustments, or corrections?

Key Personnel:
Have you identified individual programmers and other technical personnel who are in a position to inflict significant harm, or on whom your organization is unusually dependent?

Access:
Is access to your computer room, tape library, disk library, and forms storage areas denied to personnel who have no business need for access?

Scheduled Vacation:
Do employees take scheduled vacations, which can provide an opportunity to expose unauthorized practices?

Formal Procedure for Changes:
Do you use a formal procedure, which requires individual signature authorizations, for changes or modifications of software that affect systems applications?

Decoy Names:
Do your files of customers and clients contain secret decoy names and addresses so as to detect unauthorized use of those files?

Solitary Work:
As a general rule, do you prohibit anyone from working alone in the computer room?

Audit Trail:
For all major financial applications, is there a diagram and a description of an audit trail, which shows how an individual transaction can be traced through the system?

Standardized Reports of Discrepancies:
Does your internal auditing function or your security function receive standardized reports of (1) differences in cash and inventory, (2) high-dollar transactions, (3) large usages of inventory, and (4) any other unusual or inconsistent figures and activities?

Prosecution:
Would you prosecute an employee found guilty of a serious premeditated criminal act against your organization?

(Source: information from the Computer Security Institute, 8 Kane Industrial Dr., Hudson, MA 01749.)

5 SITUATIONS WHEN EXPERT SYSTEMS ARE USEFUL TO SOLVE A PROBLEM (List 870902)

Expert systems are most often used as intelligent assistants or consultants to human users. They can be used to solve routine problems, freeing the expert for more novel and interesting ones. Expert systems can also bring expertise to locations where a human expert is unavailable or to situations in which an expert's services would be very expensive. Some corporations even see expert systems as a way to collect and preserve a "corporate memory" because an expert system never retires, becomes sick, or leaves.

A problem lends itself to an expert system approach when:

A solution to the problem has so high a payoff that it warrants the development of a system: solutions are needed in the area, and other methods of obtaining them have not worked.

The problem can be solved only by an expert's knowledge rather than by utilizing a particular algorithm, which traditional programming could handle.

You have access to a willing expert who, with assistance, can formalize the knowledge needed to solve the problem. You need to interview an expert intensively in order to provide expertise to the system.

The problem doesn't necessarily have a unique answer. Expert systems work best for problems that have a number of acceptable solutions.

The problem changes rapidly (for instance, new components are continuously being introduced for computers that must be configured); or knowledge about a problem is constantly changing (as in the continuing discovery of causes and treatments of diseases); or solutions to problems are constantly changing (for example, new methods of equipment repair are being approved).

(Source: from "The Artificial Intelligence Experience: An Introduction" by Susan J. Scown, and published by Digital Equipment Corp., Maynard, MA 01754, 1985) Ω

Computing and Data Processing Newsletter

COMPUTERIZED "DETECTIVE" FOR UNIDENTIFIED CHEMICALS AVAILABLE TO SCIENTISTS WORLDWIDE

Dennis Meredith
Cornell University News Service
840 Hanshaw Rd.
Ithaca, NY 14850

A research tool that has already helped to save poison victims, catch criminals, battle pollution, rescue oil-soaked birds, classify plants and animals, fight insect pests and even aid the international trade balance will soon be available to scientists worldwide, as a result of a joint Cornell University-American Chemical Society project. It's a powerful tool for identifying unknown chemical compounds and has been used millions -- perhaps even billions -- of times by chemists over the last three decades. Its developers provide it free to anyone who needs it.

The invention is a set of two powerful computer programs called Probability-Based Matching (PBM) and Self-Training Interpretive and Retrieval System (STIRS) developed by Cornell chemist Fred W. McLafferty and his colleagues. Scientists use the PBM/STIRS programs for high-speed identification of unknown chemicals which they feed into an analytical instrument called a mass spectrometer, a device for breaking apart chemicals into component parts. By sifting through data from the spectrometer, the PBM program either matches the unknown with one of 115,000 compounds in its database; or if the compound is not in the database, the STIRS program uses "artificial intelligence" to deduce information about the size and structural parts of the unknown molecule.

Under a $318,000 grant from the National Science Foundation, Cornell and the American Chemical Society's Chemical Abstracts Service (CAS) will develop an on-line search system that will be available through the CAS Scientific and Technical Information Network, a computer database that includes some 8 million compounds. The Cornell and CAS scientists will work to increase the accuracy and speed of PBM/STIRS and to apply it to the immense CAS database, so that scientists around the world can identify unknown compounds using their computers over telephone lines and computer networks.

PBM/STIRS works like a master puzzle-solver. Mass spectrometry is basically a technique of breaking an unknown molecule into pieces and then laying out the pieces according to size. The complex pattern of pieces is usually unique for each molecule, and PBM/STIRS, in effect, figures out how the pieces might fit together to arrive at the most probable structure. Despite its immense data base size, the high-speed matching system of PBM/STIRS allows it to identify several unknown mass spectra per minute.

Most mass spectrographs are run so that they analyze compounds streaming out of a separation device called a gas chromatograph, which isolates individual components from complex mixtures of chemicals. The resulting linked instruments, called GC/MS by chemists, have become the most widespread system in science for identifying mixtures of unknown compounds. The number of compounds that stream out of such a system make a computer program to analyze them absolutely necessary, says McLafferty. "When a gas chromatograph/mass spectrometer is used to analyze, say, an extract of stomach contents or tobacco smoke or river water, the result is a huge mass of data. Inside an hour, you can get 100 unknown compounds coming out, each of which has a mass spectrum consisting of scores of separate molecular pieces in widely varying abundances."

To make sense out of this data deluge, McLafferty began in the 1950s to develop the program that became PBM and the huge database of computer-coded mass spectral data as the program's catalogue. His first system -- before even personal computers existed -- used a computer card sorter with spectra stored on punch cards. The system advanced to each new level of computer technology that became available. PBM quickly evolved into more than rote pattern-matching.

"To complicate matters, one 'peak' of material coming out of the chromatograph can

(please turn to page 24)

Vol. 36, Nos. 9-10
Sept.-Oct., 1987

Editorial Offices	**Berkeley Enterprises, Inc.** 815 Washington St. Newtonville, MA 02160 (617) 332-5453
Advertising Contact	**The Publisher** **Berkeley Enterprises, Inc.** 815 Washington St. Newtonville, MA 02160 (617) 332-5453

"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. 15, 1987: U.S.A., $24.50 for one year, $48.00 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 $24.00 per year to your subscription rate in the U.S.A., and $27.00 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 address changes 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.

computers and people

formerly *Computers and Automation*

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.

Correction

In the July-August 1987 issue, on page 27, we misspelled a name, that of Nico Bloembergen of Harvard University. We regret this error.

Front Cover Picture

The front cover shows archaeologists and scientists from IBM United Kingdom Ltd. working together at one of Europe's most famous archaeological sites, Sutton Hoo in Suffolk, England. They are recording three-dimensional coordinates of the surface features of one of Sutton Hoo's sand people, so called because the acid soil of Suffolk has eaten away the 6th and 7th century bodies and clothes, leaving only areas of discolored sand. Once the coordinates have been recorded on a personal computer, the data can be manipulated to produce a three-dimensional model of the body's shape. For more information, see page 24.

The photograph is courtesy of the Sutton Hoo Research Trust.

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

A Modern Perspective on a Computer

Edmund C. Berkeley, Editor

1. What is a computer?

A computer is a new, strange, and extraordinary kind of apparatus, now existing in many forms. It began in the 1940s; continued in giant size in the 1960s; and now exists in hundreds of sizes from smaller than a briefcase to much larger than a room full. It handles information in packages of several instructions to a million instructions; but packages of information of over 10 million instructions work badly. Packages of over 100 million instructions regularly do not work right and never work right the first time used. Input consists of data like: "Take 17. Take 22. Add them." Output consists of data like: "The sum is 39. Do the next instruction." Over 40 years the speed of handling instructions has gone from about 3 additions per second to over 100 million additions per second. More than 10,000 applications of computers have been developed; and more than 1000 kinds of work for humans have arisen or changed or vanished.

2. Is a computer magic?

Magic according to the dictionary is any extraordinary or irresistible power, influence, or charm by means of which humans can assure control over natural forces or supernatural agencies. Thus if we can make an event happen with no understanding of why or how it happens, that event is an instance of magic. But if we can make it happen again and again and again by observing related events, and soon perform the event whenever we choose to, by "causing" it to follow certain other events, then the magic disappears and vanishes.

So we can take a modern handheld calculator and put in: "17, plus (+), 22, equals (=)," and get from it "39" for an answer, and we have started to master the magic of a computer. A modern handheld calculator about the size of a thick card can receive energy from room light, accept many instructions with eight-digit numbers, and display the correct answer with lighted numerals. Because some manufacturers can produce more than ten million such calculators and sell them for less than $10 apiece, we confess these calculators to be real -- but in the primitive side of our minds, we still are convinced they are magic.

3. Does a computer think?

Thinking, in the way that a human thinks, may be any one or more of a thousand activities of the mind and body, which lead to the way that a human behaves, the actions, decisions, feelings that a complex person shows.

Some of this behavior is logical, mathematical, statistical, linguistic, artistic, communicative, or in other ways intimately related to taking in and putting out information. Such behavior is clearly able to be closely imitated by (and often excelled by) a computer. This kind of computer certainly thinks.

Other kinds of behavior of a human involve objects: repairing a faucet; mowing a lawn; driving an automobile; picking raspberries; digging a ditch; planting seeds; and much more. This kind of behavior involves far more than just information, in fact, a wide variety of skills, habits, practice, perceptions, and judgments. The skills and practices are regularly taught and changed from time to time by families, schools, businesses, and societies. When this kind of behavior is automated, the techniques of performing a process (such as filling each of a thousand pill bottles with exactly 100 pills) may be entirely different from ordinary human behavior. And the kind of behavior of a skillful plumber using three minutes to repair a leaky faucet in somebody's home will probably never be computerized.

So the sensible answer to our question is: Often yes, and often no.

Ω

Computer Intimidation and Anxiety —Part 1

Dr. John Shore
c/o Viking-Penguin Inc.
40 West 23rd St.
New York, NY 10010

"Fear, almost always, springs from ignorance."

intimidate, v. to render timid, inspire with fear; to overawe, cow...

anxiety, n. uneasiness or trouble of mind about some uncertain event...

- Oxford English Dictionary

Lasting Interest in Computers

Like some other first experiences, my first computer experience occurred during my undergraduate years. Then, my lasting interest in computers began not as the result of carefully planned Ivy League diversity, but as the result of belonging to the Yale Flying Club. I was an "active member" -- roughly translated, this meant I would rather fly than study. But I never flew as much as I wanted to, primarily because it was expensive and not included in room, board, and tuition. My parents supported my education generously, but they drew the line at flying, so most of the time I had to support the habit with my own earnings. This requirement led to a succession of odd jobs and finally to the offer of a part-time job as a research assistant in the physics department. The job was a real plum, especially since I was a physics major, so I gave up the mixed blessings of my previous position as a janitor at a nearby girls' school.

The new job was my first direct encounter with nontextbook science, and as such it had all the formative aspects of a novitiate. I was lucky in having a boss who made the experience a positive one. Dr. W. Raith was an atomic physicist with a special interest in experiments performed with beams of electrons. I helped make up the "targets" that were placed in the electron beams, plotted graphs of data from experiments, fetched articles from the library, and performed a variety of other tasks befitting my position. At one point Dr. Raith asked me whether I could learn enough computer programming to calculate the predictions of some theoretical models. Of course I said yes.

I was delighted to have someone pay me to learn about computer programming. I was also excited that Dr. Raith and others might rely on my programs. This would be analogous to designing and building equipment for their experiments, and for an undergraduate assistant to do this, rather than merely to operate the equipment, was unusual. I wish I could say that I was so honored because of my brilliance. In fact, other, more appropriate people were too busy, and Dr. Raith didn't know how to do it himself; moreover, he was uncomfortable about learning how. His reluctance was my first encounter with computer anxiety.

The Barrier of Glass

In the mid-1960s, the center of computing life at Yale was the Thomas J. Watson, Jr. Computer Center, an appropriately modern-looking building that housed several large IBM computers. The computers were accessible to all, but only visually. They sat behind a long wall of glass, where we watched them receive the attentions of full-time attendants.

In those days few people interacted with computers directly by means of computer terminals. Most people prepared their programs and data by using card-punch machines to punch holes into a series of oblong cards. Punched cards were first used to control a machine by the Frenchman Joseph Marie Jacquard. In 1806 he invented a device that attached to a loom and automated the weaving of complicated patterns, and the Jacquard loom quickly became important to the textile

industry. Punched cards were first used to aid computation by Herman Hollerith, who used them to simplify the tabulation of the 1890 U.S. Census. Later he founded a company that eventually became the IBM Corporation, and such punched cards became known almost universally as IBM cards. Today they've been largely supplanted by other media -- IBM cards aren't used in word processors and personal computers -- but they haven't disappeared entirely. For example, they're still commonly used in computer systems that issue and process payments in the form of checks. Anyone who has ever received an IRS refund or other government check has held an IBM card. The phrase "Do not fold, spindle, or mutilate" came from the IBM card and was at one time symbolic of the computer age.

Whenever I "punched up" a program at Yale's computer center, I would place the resulting deck of cards in a special tray, where it waited, squeezed between other people's card decks, until the computer exhausted its current backlog. This could take anywhere from a few minutes to a few hours. Often we would just hang around, watching through the glass.

There wasn't much to see. There was mechanical activity -- cards disappearing into machines, cards appearing from machines, tapes spinning, and paper being printed -- but all of this had to do with getting data in and out of the computer, and none of it had much to do with what was going on inside. The only visible evidence of actual computing was a shimmering array of lights, each one blinking on and off as various changes occurred within the computer. Most of us had no idea what the lights meant, but we stared at them anyway. Scenes like this were common at computer centers throughout the country. Rarely have so many stared so long at so little visible activity.

The Barrier of the Card Reading Machine

I would watch the card-reading machine, trying to tell when my deck was being read. At the critical time my eyes would shift to the shimmering lights, as I tried to imagine the computer working on my program, and then to the printer, where I could often spot the results being printed. It was like watching a magic show.

My understanding stopped at that glass wall. I understood, albeit incompletely, how my computer program posed numerical questions and defined a procedure for answering them. But when the answers came back, or when the program was rejected for one reason or another, I was always a bit surprised. I didn't have the faintest idea of how the computer did what it did. People told me that the underlying principles were simple, but I didn't believe them. I knew that, barring some malfunction, the resulting printout was strictly a function of the card deck I had submitted, but I couldn't shake my sense of mystery, even when the printout was what I expected. And when the computer rejected my program, it would accompany the rejection with a series of cryptic pronouncements that I was unable to decipher. Off I would go, seeking divination at a special desk provided by the computer center to deal with this common phenomenon. There, one more versed than I would interpret the computer's pronouncements.

The glass wall reflected our fascination and it catered to our enormous curiosity about this new technology, but it also emphasized our separation from it. We could look, but we couldn't touch. Looking longer achieved familiarity, but not insight. By emphasizing our separation and our lack of understanding, the glass wall reinforced our tendency to be intimidated.

The Barrier of Miniaturization

"It's like magic" is a common reaction to a demonstration of the computer's abilities. The urge to believe in magical and psychic powers is strong and well documented, and there's no reason to think that the urge doesn't extend to computers. Most of us, however, don't believe in magic, although we often enjoy watching magicians. We think of them as having technical skill rather than magical power, we speculate on the mechanism of deception, and we want to see how it's done. So it is with the computer.

When a machine's moving parts are visible, they help to make apparent the machine's logic. When you turn the steering wheel in your car, you rotate a shaft that moves some rods that turn the car wheels, all of which changes the direction of travel. But if you look inside a modern computer, the most activity you're likely to see is that of a fan pushing air, if that. Electronic logic replaces mechanical logic. There are no moving parts, only moving electrons. It's hard to develop intuition about moving electrons because their movements are invisible, and their effects are statistical.

Modern information processing is not only nonmechanical, it takes place on a microscopic scale. It wasn't that way at first. The first electronic computer, the ENIAC (Electronic Numerical Integrator and Computer), was housed in a room 30 by 50 feet, weighed 30 tons, and contained more than 18,000 vacuum tubes. The ENIAC was dedicated in 1946. Today, you can carry a much more powerful computer around under your arm, and you can balance its main internal units on your fingertip.

Miniaturization is a barrier to the senses and therefore a barrier to the acquisition of physical intuition. And without physical intuition, it's hard to feel comfortable about microscopic engineering. The microscopic scale of modern electronics has a lot to do with the computer's technological intimidation; it's hard to believe that so much can take place on such a small scale.

The Barrier of Microscopic Engineering

In fact, there's just as much room for microscopic engineering as there is for macroscopic engineering. The head of a pin, for example, may seem small to us, but there is enough room on it to write the entire "Encyclopaedia Britannica." And when I say "write," I don't mean in terms of some computer-readable code, I mean directly, with letters and pictures.

The "Britannica" example is from an essay by the American physicist Richard Feynman. His subject was the problem of manipulating and controlling things on a small scale, and his message was summarized in his title: "There's Plenty of Room at the Bottom." The "Britannica" example helps bring the small scale of the microscopic world into focus. Here's Feynman's explanation:

> The head of a pin is a sixteenth of an inch across. If you magnify it by 25,000 diameters, the head of the pin is then equal to the area of all the pages of the "Encyclopaedia Britannica." Therefore, all it is necessary to do is to reduce in size all the writing in the "Encyclopaedia by 25,000 times. Is that possible? The resolving power of the eye is about 1/120 of an inch -- that is roughly the diameter of one of the little dots on the fine half-tone reproductions in the "Encyclopaedia." This, when you demagnify it by 25,000 times, is still 80 angstroms in diameter -- 32 atoms across, in an ordinary metal. In other words, one of those dots still would contain in its area 1000 atoms. So, each dot can easily be adjusted in size as required by the photoengraving, and there is no question that there is enough room on the head of a pin to put all of the "Encyclopaedia Britannica."

Nature builds marvelous objects on all scales -- from the universal to the subatomic. Humans also build marvelous objects, but within a much smaller range of scales. At the top of the range are such objects as skyscrapers, oil tankers, rocket ships, and hydroelectric dams. At the other end, our finest engineering achievements have been in microelectronics, where thousands of individual miniature circuits are integrated on a small, thin slice of silicon, hence the term 'integrated circuit.' But these devices, although marvelous, are primitive in comparison to nature's achievements on the same scale. Ants and mosquitos, for example, are elaborate in structure as well as behavior. Moreover, there's still room enough for nature to make even ants and mosquitos look gigantic.

The microscopic world is unfamiliar to most of us. Our inability to see it, touch it, and manipulate it makes it hard for us to accept its reality and to appreciate its potential. But nature has led the way, and we're getting there ourselves. The smallest devices on integrated circuits today are about ten times larger than the much more complicated "T4" virus, but they're shrinking. The prospects are exciting.

Computer Printouts and Authority

I'm a skeptical reader; when I read newspapers and books, I'm quick to question their accuracy. But it's a conscious effort. In fact, I approach the printed word with a predisposition to believe. If I pick a book at random from a library's nonfiction section and read that the African spider "Arachnida Fallere" has a poisonous bite, my reaction is more to hope that I never run into one than it is to say, "Oh yeah?" and look for a footnote. When I'm personally familiar with something that I read about in "The Washington Post," I usually notice inaccuracies. But that doesn't stop me from opening the "Post" every morning, fully prepared to believe what I read. Power to the printed.

This power extends to the computer printout. When my checkbook disagrees with my bank statement, I assume that I goofed and I start looking for my error. Some people don't even bother with a monthly reconcilia-

tion; they just rely on their bank statement. Of course, most bank's computers have a good track record, and this encourages our reliance. But I think there's more to it than that.

To an extent, I think the urge to believe the printed word comes from the role of the printed word in education. Indeed, since much of what we've all learned came from books, we tend to acquire the habit of belief early in our lives. I also think that the act of publication is seen as mute testimony of accuracy. Considerable effort and expense is often involved, suggesting that someone other than the author has judged the information to be worthy of printing. In a free society accuracy is also encouraged by legal penalties for fraud and libel. Whatever the reasons, the urge to believe the printed word is strong and deep-seated. I had it long before I ever heard of computers or saw a computer printout.

In Machines We Trust

Our industrial tradition is one that glorifies the mechanical. And today, despite the modern vogue for the handmade, our society remains infatuated with machines. Machines transport people, goods, and information. Clocks, traffic lights, and telephones shape our daily patterns. In our factories, in our offices, in our wheat fields, and in our kitchens, we rely gladly on every kind of labor-saving device. We may complain about quality control and service, but we routinely drive cars, board airplanes, turn on microwave ovens, and submit to machines at the dentist. How often have you been on a malfunctioning elevator? How often do you hear about one? Not only do we rely on machines, we have a firmly ingrained habit of trusting them.

This habit, which predisposes us to trust computers, is reinforced by several factors: The computer isn't just a machine, it's a machine that communicates with us. Moreover, it often communicates in terms of intimidating technical jargon, and it can communicate with incredible speed.

Nothing symbolizes our trust in computers better than the printout. I remember how often we stood in front of the glass wall at Yale, staring at the printer. It was exciting to watch it come suddenly to life whenever the computer was ready with results. I think it was the lack of human intervention that made the scene so captivating. And the speed -- even in those days the printers spewed results at remarkable rates. Today I'm still drawn to the printer, and I'm not alone. People like to watch computers print answers. And people are prepared to believe those answers.

The computer is a printing automaton. In the computer printout the innate power of the printed word is magnified by our glorification of machines and our trust in them. Machines have always been involved in printing. But before the computer, people didn't perceive machines as participating in the decision of what to print.

In fact, computers don't decide what to print, although we often speak as if they do. Computer printouts are determined by input data and by computer programs. If the input data is wrong, so will be the output -- "garbage in, garbage out." Furthermore, correct outputs also depend on correct computer programs, and -- as I'll discuss more in later text -- the typical large computer program is considerably more likely to have a major, crash-resulting flaw than is the typical car, airplane, or elevator. The computer may be the ultimate machine, but today it's less trustworthy than many of its predecessors.

Appraising Printouts

Feelings of intimidation can be an exaggerated form of respect. In this sense, the typical computer printout is less worthy of your respect than traditionally printed material. A principal reason is that computer printouts are easy to produce and easy to revise. Traditional printing is more difficult -- it takes longer, its products are harder to revise, and it usually requires the coordinated activities of several individuals or organizations. These difficulties have encouraged the use of editorial review and other institutional controls that are characteristic of formal publications. Computers can make it easier, quicker, and cheaper to print formal publications; individuals can do what was once practical only for organizations. The computer has reduced the economic and practical importance of institutional controls, and so the controls are often relaxed. Earlier I argued that our predisposition to believe the printed word arises in part from the mute testimony of the publishing act. In the computer age the value of that testimony is disappearing.

Computer printouts are not only getting easier to produce, they're getting harder to recognize. The result isn't always beneficial. One example, which affects me person-

(please turn to page 26)

Software Development Systems —Part 2

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."

New Languages to Express Domain Knowledge

We are starting to see some highly specialized languages that incorporate a great deal of information about an application. Languages for the control of a particular piece of equipment are of this sort, as are the program generator packages that are based on an underlying model of some specific application. These languages presently cover only a very small part of the total range of applications, but research is under way to make it easier to produce application-specific languages.

An even smaller intersection exists between software engineering knowledge and domain-specific knowledge. We have very few principles in software engineering that specifically address how to build applications of a particular type (for example, hotel registration systems). To some extent, that is OK since we do have some specifics that cover entire classes of applications (for example, real-time systems or database systems). However, this knowledge is woefully inadequate at present.

As you might expect, the smallest overlap between classes of development information occurs when we try to find information in each class that is relevant to a specific system. This intersection is absolutely critical to the successful creation of software since it conceptually contains all the information necessary to bring the desired system into being.

There are some strategic implications of this state of affairs. Creating software representations (new languages) that do not help us to better utilize domain knowledge and/or software engineering knowledge belongs in the area of pure research, that is, exploring the implications of language design principles for their own sake. That is quite important for the long term and certainly should be supported. It is important to recognize, however, that to gain more immediate benefits it may be best to focus on new languages that can better help utilize the domain and development knowledge we now have.

Control of Information

Control of information is at the heart of controlling development -- and representation is at the heart of information control.

In terms of pragmatic software development, it is important to manage the balance between the different forms of information in the development environment. A group that is heavy on software engineering skills but does not have effective access to domain-specific (application) knowledge will have great difficulty building the systems needed by the customers. On the other hand, a group that thoroughly understands an application area but that is short on software engineering know-how stands in danger of foundering on the complexity of the software they try to create. Underlying both is the necessity for effective ways of representing the information so that the overall process can be managed.

Information is the essence of software. If it is not managed correctly, it will rapidly become confused and lost, with predictable impact on the software.

In reality, this often happens. The situation is similar to that in hospitals 100 years ago. As discoveries were made about the role of various micro organisms, people realized that many of the things they had

been doing in hospitals were very dangerous; as a result, major attention was suddenly paid to practices (such as the sterilization of instruments) which previously had been ignored. We are just beginning to appreciate more fully the role of many kinds of information on the process of producing executable software, and thus to treat it with more respect.

At present there is a poor balance between naturalness and formality in handling development information. Many technical personnel focus only on the executable forms of software, leading to problems. For example, much good information about the structural aspects of a piece of software is lost during development. Initially, this may not pose a problem, but eventually it usually does since this is precisely the information that is needed to help preserve the integrity of the software during evolution (typically called maintenance, although it incorporates three kinds of change: repair, adaptation, and enhancement). This is similar to an organization that divides itself into departments and then loses all information about which employees belong to which departments; reorganization will be very difficult.

Belady and Lehman, in their pioneering studies, found that structural information (definition of subunits such as modules and the interfaces between them) was essential to maintaining technical and managerial control of a software system over time as it undergoes change. In spite of a wave of techniques incorporating "structure" in their names (programming, design, analysis, testing), it is still common to find that after initial development, no coherent record of the system's internal structure is maintained.

Management, on the other hand, too often assumes it is all technical information to be left to the technicians to be handled; again, valuable information for management, such as the reasons why a particular feature is requested by the customer, may be lost. Their response, once the importance of information is realized, is sometimes to mandate that everything be saved -- leading to a very "formal" system, laden with paperwork and attendant problems.

We should strive to be more relaxed and intuitive about the information in the development environment, while still preserving in usable form relevant parts of it. For the present, however, while we are still discovering which information is most important and learning how to deal with it, we must be more explicit. To that end, let's look at software, in the general sense, and see what happens to it over time.

Forms of Software

You are probably familiar, at least vaguely, with many of the forms software takes:

- Programs in machine language
- Programs in higher-level language
- Specifications
- Needs statements
- Requirements
- Architectural designs
- Detailed designs
- Data formats
- Collections of programs
- Programs being tested
- Finished programs
- Systems in use for production

as well as some of the other forms of information which I include under the generic term "software":

- Analysis of requirements
- User documentation
- Maintenance documentation
- Change requests
- Modification specifications
- Error reports
- Performance measurements

and so on!

I often encounter one of several reactions when I start to discuss the forms of software: People who are deeply involved in the technical development process want to argue for their particular definition of some of these elements; those not so deeply involved want to be given an explicit definition of each form and be told which they should use; some who are involved, but not directly, in the technical development, couldn't care less and want to get on to consideration of what they consider to be the critical issues. It is like analyzing an election; everyone has a different viewpoint.

All of these viewpoints are understandable and, in their place, appropriate. Here

I want to encourage a different viewpoint for the purposes of improving your understanding of the underlying principles of good software development: that of an observer of the development process who is trying to see what forms this slippery substance called software takes during its journey from idea to production system.

This viewpoint can very quickly lead to arguments about the relative merits of different sequences of steps that are taken in developing software (usually labeled as arguments about the best lifecycle to use). We will come back to some of that later, but again, let's delay the arguments about specific approaches until we understand a little better the underlying nature of what we are dealing with.

The Life Cycle of Software

The concept of a lifecycle has been adopted in the world of software to indicate the stages that software goes through. As in biology, from which we borrow the term, a piece of executable software goes through a set of stages that are more or less dictated by the nature of the animal (and to some extent by the techniques we have for bringing the finished product into existence). Roughly, software goes through the following stages:

Concept development (definition)

Technical creation (development)

Product for use (operation)

Object of further development (evolution)

The methods, tools, and techniques that are used have little impact on this basic, overriding sequence. We must start with an idea of what the system will do, then we create it, use it, and, quite likely, modify it. In each form of the software, we have a representation of the final object which will be loaded into some hardware to create the desired behavior.

It is important to understand that there are variations on this basic sequence that may be extremely important. Just as children engage in limited adult activities so that they can learn, it is often critical that software be exposed to some operational stresses while it is being developed in order that we can learn at the earliest possible moment whether it will stand up to its ultimate "adulthood" or not. We understand more clearly now that software development cannot always have a completely strict progression, but may need to cycle through stages in an iterative fashion. Software prototyping, for example, is one expression of this basic concept.

At this point, I am trying to keep from becoming too specific (I know this is frustrating to some of you, but as in the technical design of software it sometimes pays not to rush too rapidly to the final point) in order to provide a broad perspective on the nature of software. I find that the following description (with examples) of the forms of software is useful:

1. Development Prologues
   - Needs statements
   - Analyses of needs
   - Requirements statements
   - Analyses of requirements
   - Data element definitions
   - Functional specifications
2. Technical System Descriptions
   - Technical specifications
   - Architectural designs
   - Detailed designs
   - Database structures
   - Data descriptions
   - Executable programs
3. System Aggregations
   - Collections of programs
   - Systems of programs
   - Interface definitions
   - Integrated hardware/software systems
   - Integrated human/hardware/software systems
4. Installed Systems
   - Reference (baseline) systems
   - Versions
   - Production systems
5. Performance Data
   - Efficiency measurements
   - Error reports
   - Effectiveness ratings
   - User ratings

Let me repeat: My purpose here is not to provide absolute, technically perfect definitions of software artifacts. Rather, I am trying to provide some abstractions that will help you gain some perspective on what is going on around you. Let's look briefly at each of these categories.

Development Prologues

It is clear that something comes before the technical development of software -- a memo from the boss, a concept for an application sketched on the back of a napkin, a

detailed statement of the functions that are needed in a system, analysis of the needs and stated requirements, and so on. The general nature of this information is that it focuses on the functionality of the system, its usage in some instrumental setting, its relationship to other systems -- in short, looking away from the underlying physical hardware (one of the things that goes awry in development is that technical descriptions are sometimes mixed in with this kind of information before a clear understanding of the external aspects of the system is gained). I have labeled this "development prologues" to emphasize that it is information that comes before the technical development. Yet, in all of it we can glimpse some aspect of the eventual system. Development prologues are like the editorials and speeches and articles that precede some governmental action, such as passage of a new law.

Technical System Descriptions

"Technical system descriptions" are the familiar artifacts found in designers' offices, on programmers' desks, scribbled on blackboards (or, the programmer's favorite, whiteboards, which provide the extra dimension of easy coloring). There shouldn't be much of a problem understanding this class. It is worth noting the overlap that exists in practice between development prologues and technical descriptions in the area of specifications. Indeed, a good set of specifications is precisely the bridge between the outward-looking development prologues and the inward-looking technical descriptions since they can be considered both at the same time. Continuing the analogy, technical system descriptions are the laws and regulations.

System Aggregations

"System aggregations" are technical artifacts, too, of course, but I have separated them to emphasize the change in focus. Up to the point where we have individual components (running programs in the case of software) our focus is on those individual pieces. In building large systems, however, the battle is barely begun once the pieces exist because then we must integrate them into a working system. This activity not only takes a large amount of effort in most cases, but it has fundamentally different concerns. Hence I think it is useful to look at the aggregations as a different type of object. In the analogy, system descriptions are the collections of laws and regulations pertaining to some subject.

Installed Systems

Once we have a system built and ready for use, the focus again changes, so I have shown a fourth class of system objects: "installed systems." Now we are concerned with entire systems, not individual pieces but the aggregations that make up the total system. We treat them differently so again it is useful to separate them. Installed systems correspond to the laws actually being used and interpreted.

Performance Data

The last class I have shown, "performance data," may be the hardest to accept, since I have now transcended artifacts which we can all recognize as in some way representing at least a part of an executable system and entered the realm of abstractions. Instead of descriptions of systems, this class contains measurements and other data that characterize the system in use. In the analogy these are the statistics and observations collected on the results of using the law.

Software Is NOT Easy to Change!

One of the enduring siren songs of software entices us with the message that it is easy to change. Countless managers and unwitting programmer accomplices have looked at a working piece of software and been seduced into believing that they could easily make some changes to improve it or transform it into something else. Little did they realize the agony, frustration, and destruction of good relationships (for example, with their customers) that lay behind that seemingly benign and simple encounter!

This two-faced situation -- on the one hand lines of code are very easy to change, while on the other systems can be incredibly difficult to alter successfully -- comes about because of the underlying fact that software is a system. It is the thousands of interwoven dependencies and interconnections that pull us down into the tarpit, not the individual pieces.

Consider a modern manufacturing plant. A complex system by anyone's standards, physical facilities, supply deliveries, power, labor rules, and hundreds of other things (both hard and soft) must work together properly to create a productive environment. Yet almost none of these elements are easy to change. Power systems are expensive and require long lead times to build, alteration

of labor rules may require years of bargaining, suppliers may be scarce for certain parts, and so on. Consequently, we are not so tempted to change the manufacturing system. (Indeed, this difficulty of change has probably been perceived as harder than it is by some, hastening the decline of certain industries.) The underlying mechanisms of the system are hard to change and hence we don't lightly undertake systemic changes.

Oh, that we had less pliability in software!

Hidden Relationships

Technically, it is the unforeseen consequences and hidden relationships that kill us in software. We think that a minor change in the screen format of a presentation will improve its readability (undoubtedly true), so we ask the programming staff to make the change (they have indicated it is only a minor change to one module of code). The change is successfully made.

Unfortunately, the change involved expanding slightly a table used by the module, pushing the system size above the scheduling limit for small programs, meaning the program now gets significantly less rapid service, resulting in a response time of the program to requests from the user at the terminal that is now nearly three times as long! Since the program is used to generate information to support a telephone marketing representative while on the phone, the system is now useless.

And, of course, the programmer making the change finished at 8 P.M. on Friday night, deciding to leave documentation until he returned from the two-week camping trip to the wilderness on which he left the next morning at 5 A.M. Thus a seemingly minor change in a program led to a major change in the performance characteristics of the program resulting in several weeks lost productivity. This is a rather obvious example of an unintended design consequence -- one that can be readily seen and understood. Others are not so evident.

Coherent Conceptual Basis

In building software we do not yet have the coherent conceptual basis that we have in other pragmatic disciplines. Physics underlies electrical and mechanical engineering, giving us a way of predicting consequences and understanding relationships. Chemistry underlies chemical engineering. Biology supports medicine. Because software is all man-made and interpreted by finite and deterministic machines (computers), we believe that we can ultimately understand the underlying science of computers and hence be better able to deal with consequences and connections. At present, though, we are much closer to medical doctors (who have only an imperfect understanding of the underlying nature of the human body) in having to prescribe actions with only a partial understanding of the results to be expected. Even worse, we have much less experience than the medical profession!

The siren song of software flexibility has seduced many well-meaning professionals.

Web of Connections

As though our inability in many situations to understand design consequences and system linkages were not bad enough, there is another dimension that causes perhaps even more trouble and expense in the long run. That is the web of connections between the different aspects (or forms) of software discussed above, the chain of representations extending from needs statement to specifications to design to code to test results. The technical problem is one commonly called the design-updating problem. What usually happens is that even when a careful chain of system representations has been built, once the system exists, changes will be made to the programs without reflecting those changes back into changes in the design and the specifications. The result: The more abstract representations of the system are quickly made useless, so that future changes to the system must be made using only the program listings (or an equivalently low-level representation). In building construction, this would be like building without plans or standards and keeping no record of where pipes and wires were installed after the initial construction so that future changes to the building required walls to be torn out simply to find out where the wires were.

What are the implications of the reality of software being hard to change? There are two, in general terms. First, build the system as though it will last forever, enduring many changes along the way. Second, control of and changes to any system must be made very carefully. Let's look a bit more at the first implication.

Old Code Never Dies

This is a reality that many people and organizations learned years ago. Yet be-

cause the average length of experience of people in the field seems to be dropping (due to the rapid expansion of the use of computers and hence of people's involvement in creating software) it is a lesson that people constantly learn anew the hard way!

Software lives forever for any number of reasons, good and bad. The most obvious and pervasive reason is that once built and in productive usage, there is inertia that must be overcome if it is to be changed or replaced. The users like it. The boss likes it. The customers like it. It doesn't matter that it is slow, prone to errors, needs to be expanded to take care of new functions, and so on. The software has become a part of their lives and all they want is that it be fixed up a bit -- certainly not replaced or changed in any major way!

The relationship between software and the people around it (users, customers, managers, operators, and so on) is not unlike the relationship between people that find themselves associated for historical reasons (e.g., in a marriage, working situation, neighbors) which may have little bearing on their present situation. It takes a tremendous amount of effort to change the situation and break the relationship to the software! The result: The software lives on and on and ...

There are often economic reasons why software doesn't die, beyond those stemming from its intimate role in the production process. Software of any magnitude costs quite a bit to create in the first place (and, worse, it usually can't be considered a capital cost). If it has been used very much, then quite likely considerable money has been spent on it over the years to repair, adapt, and enhance it with new functions; it is not uncommon to find that three or four times the original development cost has been spent in this way. That turns out to be a large investment, generating the expected inertia on getting rid of it (even if the users agree that it is time to replace it).

Finally, it is not uncommon to build a piece of software for a "quick" application, one that supposedly will last for a limited time after which the software will be thrown away, only to find that the application is so productive that the decision is made to continue it -- together with the use of the quick (and dirty) software!

The moral is: You must expect that software will never die. Build it as though it will continue far beyond your own retirement, will be adapted to run on new equipment, enhanced with new functions, repaired to take care of those pesky special cases, and generally outlive those that created it. There are obvious counterexamples -- truly one-shot situations, applications for equipment that is about to die, functions that are eventually outmoded -- but it is better to err on the side of too much preparation for the future rather than too little.

Software, like unwanted relatives, is difficult to get rid of.

Software As a Product

The history of computing has fostered the view that programs are highly mathematical in nature, created by genius mathematicians to solve esoteric problems understood only by them and a few others. Indeed, that was the nature of many early programs and the people that created them. As computers were applied to a wider variety of tasks, people started to realize that in fact computers are very general symbol processors, not just number crunchers. That view has now become dominant because of the large amount of non-numerical (or, at least, involving only very simple arithmetic) work that computers do and because of the advent of the personal computer used by millions of people for very ordinary tasks.

Another view, clearly a corollary of the first, was that software was not a product in the sense that one described it, packaged it, worried about inventory control and derivative models, marketed it, sold it, maintained it, and removed it from distribution when outdated. That view, too, has changed radically in the past 20 years, although the change has not been as pervasive nor as well heeded as the change in view regarding the essential nature of a computer.

Starting with the first "unbundling" of software from the price of the hardware in the late 1960s, software has gradually come to be seen as a product; in the past five years, the explosive growth of the personal computer field has made it abundantly clear to most (although not all) people that software can indeed be treated as a product.

Interestingly, many of the "software millionaires" are not old-line professional programmers, but rather people who quickly grasped in various contexts the need for a particular software product, created it, packaged it, and got it to the market first.

Indeed, many of the most successful software products are not very sophisticated in technical terms (there is a lesson there, too).

Creating the idea for a product is not the same as building it. Neither is marketing and selling it the same as building it. In building a real product, one must pay careful attention to serviceability, reliability, customer tastes, cost of producing it, and so on. These aspects of what it means to make a product are only starting to be understood and taken into account by the development community. I think this is why many of the biggest successes in the software product field have been created by nonprofessionals.

At times the product ignorance of some technical people is actually a blessing. When building the software itself, it is important to focus on the technical characteristics and find the best technical solution within the constraints of the product parameters. If everyone is focusing on the product parameters, it may be more difficult to achieve a satisfactory technical solution.

Perhaps the most important thing is to keep the product and technical aspects of software in balance. All software is ultimately a product for someone else (perhaps just the builder using it in a different context), so it is important to deal appropriately with the product aspects. On the other hand, all packaging and no content is even worse than all content and no packaging (which can usually be more easily remedied).

Viewing software as a product quickly leads to many implications for the development process: How we are to determine requirements from an anonymous set of users, how the design and implementation process can be speeded up, how we handle changes after it is released, and so on. Some of these issues will be discussed later in the context of individual parts of the development system.

Treat software as a product as well as a technical object.

In this context, it is important to recognize one of the main product characteristics of software, namely, that it is essentially packaged knowledge.

Software Is an Embodiment of Knowledge

A view of software that I find especially appealing (perhaps because of its grandiose overtones, but, I prefer to think, because of its all-encompassing nature) is that software is an embodiment of knowledge. Any program certainly contains a large amount of information about a process and the data that are relevant to carrying it out. Programs also contain structural knowledge that indicates the relationship between different processes (programs) and their associated data, knowledge about the relationships among classes of data, and other information as well.

If programs, the end result of the chain of representations that we call software, contain knowledge about the outside world, then certainly the earlier versions -- the designs and specifications and requirements -- also contain knowledge. Indeed, one of the realities of software is that these forms of the system often contain valuable information which is lost before we get to the executable versions!

Whatever you may think of the grandiloquence of this view, it is well worth your pondering. If software is knowledge (I am not wasting many words to convince you of what seems so obvious), then the argument for taking great care to capture and manage properly the information that exists in the software development environment certainly takes on a new importance. Likewise, it adds weight to the plea, often heard, that more attention should be paid to the activities that come before the actual coding of programs. Those activities are producing something at least as valuable as plans for programs -- knowledge. This is one of the primary motivators for software reusability.

Programming -- in the narrow sense of laying down sets of instructions that tell a machine precisely what to do -- is comparable to writing a research report. It produces a representation of knowledge that has been acquired before the writing began.

In this context, it seems clear that what we are doing in the early stages of software development is uncovering and organizing knowledge. First, and most importantly for the larger picture, we are producing knowledge about the application; then, during design, we move on to producing knowledge about how to deal with the application with the computer.

The interaction between these two discovery processes (and the methods used) is complex and not well understood. Even less clear is the relationship between these pro-

(please turn to page 27)

The Current State of Desktop Publishing

Ann M. D
Highland A
P.O. Box
Dunn Lor

"Almost anything that is typeset, and many things that are not yet typeset, because the task is too difficult or costly, may benefit from using desktop publishing tools."

Computer-Aided Publishing Defined by Equipment

It has been said that being in computer-aided publishing is as undefined as being "in transportation." With transportation, one can get to California from Virginia on a moped or on a Boeing 747. The end result is the same but the time and difficulty of getting there is significantly different. This is also true of publishing where the resulting page or document might look the same but the blood, sweat, and tears that went into producing it might vary considerably according to the capabilities of the system used.

Desktop, as well as high performance publishing systems, is a collective market that analysts predict will top $35 billion by 1990. What can you do with these systems? Newsletters, brochures, newspapers, business forms, price lists, catalogs, manuals, books, technical documents ... Almost anything that is typeset, and many things that are not yet typeset, because the task is too difficult or costly, may benefit from using desktop publishing tools. This article, which focuses on desktop or microcomputer-aided publishing, attempts to define what can be done, types of equipment available, end user benefits, systems considerations, future trends and other aspects of interest to automated document specialists. It is important to note that this is an extremely fast developing field so that any article on desktop publishing published in a bimonthly magazine is likely to be somewhat dated by the time it reaches the reader.

For many years, publishing has been done the traditional way by "pasting up" typeset text with graphics produced separately into attractive page formats. Around 1975, the first typesetting systems based on mainframes were available and, by 1980, they were controlled by minicomputers with "front-end" systems such as Penta and Atex. The systems were very expensive and suitable only for a highly trained professional's use. In 1985, a product and price synergy occurred that made desktop publishing feasible: the Canon printer with font scaling and graphics integration at a dramatic price drop ($5000 vs. $20,000 for previous laser printers), the Apple Macintosh, a graphics-oriented PC, and easy-to-use page layout software. These products created a usable publishing system for under $12,000.

Desktop Publishing

Suppliers tend to define desktop publishing (DTP) in terms of what they already sell, so to the word processing vendor, DTP is sophisticated word processing; to the printer vendor it may be the whole range of printer output except for phototypesetters; to the graphics supplier, it might be a method to package and manipulate graphics. In this article, it is defined as the whole process on a microcomputer where the text and graphics electronic files (input in several ways described below) are manipulated by the DTP software to compose and paginate pages which are then output in different ways according to the user's quality needs. The key difference from word processing then is the relatively sophisticated software that will set typographic characters, edit graphics, format pages, and interface with sophisticated printers and phototypesetters to produce the font scaling and graphics integration. Desktop publishing systems are microcomputer-based and are in the general price range of $10,000-20,000. Workstation publishing systems such as Interleaf and Kodak KEEPS are minicomputer-based and range in price from $30,000 and up. Even though many of the

same results can be produced from both, the workstation is appropriate for the serious user who demands high resolution graphics, speed, performance, and the ease of use that comes from working with a comprehensive software solution rather than a diverse set of programs. In many cases, a firm may have both, or even integrate with the mainframe depending on needs of individual departments.

Applications and Benefits

The leading users are those with heavy document needs such as aerospace, electronics, manufacturing and government. Typically these users require frequent updates, the ability to manipulate computer-aided design files, long documents and often the power of a workstation. Examples are the department at Digital Equipment Corp. which produces technical manuals for network and communications products using workstations; or, software publishers such as Handle Technologies in Houston who produces product user manuals on equipment from NBI Inc. of Houston, TX.

In the middle are many firms that previously sent their work outside to a typesetter, or used word processing copy, who can now cost justify publishing their own on desktop systems. Examples include associations producing member newsletters, and consulting firms such as Booz-Allen & Hamilton that have a competitive edge by producing proposals and publications that are more readable and attractive.

Low cost systems mean that even the smallest businesses are experimenting; for example, an antique dealer who produces a newsletter with scanned in pictures of her antiques to increase sales. A recent but growing development is quick print shops who offer their customers design services or let the client come in and do their own design -- a perfect solution for those who either cannot afford a system or expect to use it infrequently.

The benefits cited most often are cost and time savings, convenience and control, improved appearance, and security. Approximately 6-10% of corporate budgets is spent on printing and publishing of ancillary publications such as product or training manuals. According to David Boucher, President of Interleaf, Inc. of Cambridge, MA, it is possible to save one-half or more of the cost by automating. Time savings, convenience, security, and control are all benefits related to the fact that the work is done in-house. Previously production people spent a great deal of time preparing and proofing text, proofing again when it was typeset, and again when graphics were added, and again ad nauseum. There was little control over the schedule and heaven forbid if changes were required! With DTP, it is possible to typeset and layout documents in your office, see the results immediately, and make corrections at no extra charge. Appearance will be improved over word processed documents due to the addition of graphics, improved layouts, and better font selection. However, if material has been output previously to a phototypesetter with a high resolution (1200 dpi or more) and is now output from a laser printer at "near letter quality" of 300 dpi resolution -- some readers may notice the drop in quality. Many readers will not notice the quality change, however, unless the two outputs are compared side by side.

As DTP gives people more artistic control, however, some users warn that this could be a mixed blessing. The ease in which layouts are produced and designs quickly changed, could lead to "automated ugliness." One still needs to use solid design principles and maintain control over the final output.

Considerations in Selecting a System

Users' needs range from the casual user who might be satisfied with a simple add-on font package for his word processor to extremely sophisticated combinations of text and graphics that require powerful systems to implement. Therefore, it is critical to understand exactly what your needs are. It is also important to realize that DTP is not one technology, rather it is the intersection of text processing, graphics, page composition, and laser printing, all of which must work well together. Each of these technologies matured individually and tends to have incompatible standards. Here are some questions to ask yourself.

How is publishing done now? Are documents primarily long, under 10 pages, or both? (Software is often oriented one way.) How will input be accomplished? If it is word processing, is the software compatible with the system or will the word processing people have to be retrained on new software? Will one person be controlling the document look or will a number of former word processing people be trained? In the latter case, it is wise to have a "document standards" manual so all output conforms to a company look.

Some software incorporates kerning, leading, hyphenation dictionaries and other typographical features -- are the users accustomed to making these decisions or should you choose a more intuitive layout package? What type of graphics do you need to incorporate: photographs, line art, CAD (computer-aided design) files, other? How will the system deal with them? What about output -- does your audience demand high resolution? In this case, be sure you can communicate with a phototypesetter. Can you see the entire page on the screen before you print it to be sure it is really what is desired?

Does the software you have chosen support the peripherals such as the printer and scanner you need? Do all the various components work together smoothly when hooked up? Can you telecommunicate to other locations such as service bureaus with typesetting equipment? Does it work in a networking environment so multiple writers can transmit word processed material to the layout person? How much training is required? Is the system expandable as new technology arrives? How will the system be serviced? Is it compatible with already owned equipment?

Categories of DTP Systems

There are several categories for publishing with mini- or micro-computers (note that any price ranges listed should be considered relative since prices are rapidly changing):

1. The minicomputer-based workstation which is for the "serious user" with heavy graphics, long documents, and high performance needs. These systems typically range in price from $30-100,000 depending on the configuration. Interleaf, Inc., one of the early market leaders, has been very successful by offering its software on several different CPUs including Apollo, Sun, Microvax, and the IBM/RT. Numerous other companies such as Qubix Graphics Systems, Caddex Corp., and Autotrol offer solutions for users with more specialized needs. Due to space limitations, workstations will only receive this brief mention; however, many of the considerations are similar to those below.

2. The microcomputer which offers several levels of solutions:

- The "high road" (expensive, polished route for professional artists and typographers) includes page makeup programs for PCs that can send output to phototypesetting systems or control the typesetting equipment directly. Without the typesetting equipment, these systems would range around $15,000 and up.

- The "middle road" (not so expensive but still high quality) includes the Apple Macintosh or IBM PC with page layout software and a Postscript (or other page description language) -- driven laser printer. Systems price range from $10,000-20,000.

- The "economy route" (least expensive but varying quality) includes the less expensive laser and dot matrix printers with PCs or Macs using word processing and graphics programs, with special software to control printers.

The rest of this article will focus on the "middle road" since that is of the broadest interest. But for now, which route should you take? Decide based on your needs. The high road is the one for many advertising agencies, graphics houses and publishers. The middle road is very attractive for large and small businesses, newsletter, magazine and book publishers. The low end is appropriate for school newsletters, church and club bulletins, reports, and correspondence.

The other hardware and software you need depends on your application. Most writers need enough hard disk storage to accommodate files, enough memory (usually 640K) and a graphics card with a high resolution black and white monitor for page makeup. Software varies widely in price and functionality.

Methods for Creating and Entering Data

Nearly all computer publishing products fall into the categories of creating and entering data, putting the pages together, and getting output. The following is an overview of these areas along with some examples of current products on the market or recently announced.

The basic methods for creating and entering data are:

- Word Processing
- Painting and Drawing
- Optical character recognition (OCR) Devices, Digitizers, Tablets, and Scanners

Word processing software has evolved into an established useful tool. The speed of in-

put is limited only to the speed at which the user can type. Editing and revising is simple and quick. Some of the higher end word processing programs have some elements such as two column output that in the past have been associated only with page layout software. The most recent trend has been for word processing software to integrate more easily with graphics packages. Prices are in the $300-700 range.

Painting and drawing software packages for creating images are available for nearly every type of personal computer. Most recommend the use of a "mouse" pointing device rather than the keyboard so the user can freehand draw similar to drawing on paper. Paint programs let users change each picture element (pixel) of a screen to make subtle changes in an image. Other programs make it easy to draw lines, boxes, and shapes with precise angles. Some even incorporate large "clip art" libraries so the nonartist can produce gratifying results simply by selecting. Once drawn, an image can be duplicated on the screen many times, and each copy can be enlarged, reduced, stretched, or compressed. Prices are generally in the $100-700 range depending on capability.

Retyping already typed material is now unnecessary, thanks to OCR devices that cost as little as $3000. Some, however, may not accept typeset material or may only read certain typefaces. Graphics scanners can read in line art easily and the more sophisticated ones can handle gray tones such as in photographs. In the under $5000 price range, the efficient OCR devices do not handle text efficiently. Although this technology is making tremendous strides, the user is advised to try it before buying.

Various companies make drawing pads and light pens for tracing images, or drawing images freehand for storage in the computer. Image digitizers such as the MacVision digitizer from Koala Technologies, which takes images from a conventional video camera and reproduces them on the screen, are available.

Putting the Pages Together

Page makeup software utilizes the text and graphics files created separately and allows the user to layout the page in an attractive design. Most allow a variety of fonts and, at least, simple editing of the text and graphics. Macintosh has been the leader but more software for the IBM PC is announced monthly.

The programs fall roughly into two types: those that are WYSIWYG and those that are not. WYSIWYG means "what you see is what you get" and is used to describe systems where you can instantly see on the display any changes you make. Examples are Ventura Publisher from Xerox Corp., Pagemaker from Aldus Corp., and Frontpage from Studio Software Corp. This type of software is ideal for the shorter document although some can also handle long (over 10 page) documents. WYSIWYG allows you to try it over until the page is right and then send it to the printer. WYSIWYG is economical because you don't waste printing time and paper and you can usually catch mistakes on the screen. On the other hand, micro-based WYSIWYG can be a time-consuming process especially with multipage documents so be sure that the system chosen can perform to your needs, or consider the exceptionally fast WYSIWYG solutions like Interleaf, Inc. workstations. The non-WYSIWYG (or code intensive) systems are designed for professional typographers who already know about typesetting features and like the control this method gives. These are harder to learn but can be more efficient for longer documents where the page design is essentially the same. In most cases, this type of program is not able to handle graphics other than simple boxes where graphics can be pasted in. Users with graphics needs in long documents should definitely consider workstations.

Getting Output

There are many kinds of printers ranging from the least expensive dot matrix to the most expensive page printers. The dot matrix printers are capable of near letter quality text and graphics, whereas the laser printers can emulate the letter quality text and handle 300 dpi resolution graphics. Any program that can print to Postscript printers such as the Apple Laserwriter can also be output to Postscript-type phototypesetters such as the Allied Linotype. Some software such as that from Studio Software Corp. and Interleaf, Inc. is capable of outputting to other types of high resolution typesetters (1200 dpi or more). In this case, most users would want to have a laser printer to proof the page before typesetting.

The choice of printer primarily depends on your budget and the quality of publication needed. For example, if you want the best possible laser printer resolution on graphics and font flexibility, you should consider something like the LaserWriter (about $5000) versus the less expensive la-

(please turn to page 26)

Knowledge-Based System Designed by Purdue Univ. Helps Grain Farmers

Sue Metzler
c/o "Artificial Intelligence Letter"
Data Systems Group
Texas Instruments, Inc.
P.O. Box 2909 M/S 2222
Austin, TX 78769

"A farmer with 100 acres in corn yielding 100 bushels an acre could make $10,000 more by making the best decision instead of the worst decision."

Converting Agricultural Research into Practical Results

Grain farmers are receiving help of a meaningful and lasting kind from Purdue University, long a leader in converting agricultural research results into practical tools for farmers. The help this time is a knowledge-based [artificial intelligence] system that helps farmers select the best way to market their grain.

To the majority of us, who market our services for wages or salaries, the grain farmer's need to select a marketing method every year for every crop seems alien. He's faced with selecting among more than a dozen basic alternatives, and making the best choice is critical. For example, Number 2 Yellow Corn is selling at the moment for about $2.30 a bushel. The difference between making the poorest choice and the best choice could mean as much as $1.00 per bushel in net income to the farmer!

So a farmer with 100 acres in corn, yielding 100 bushels an acre, could make $10,000 a year more by making the best decision instead of the worst decision about marketing his corn.

The knowledge-based system, based on Purdue's years of research in grain price action, was produced with the Texas Instruments (TI) Personal Consultant™ expert system development tool. It can be delivered on any of several personal computers. This is critical to the widespread use of the system. Many farmers own personal computers and use them routinely in farm operations; those who don't can usually find easy access to one through their local farm cooperatives or agricultural extension services.

Ronald Thieme, a graduate instructor in research in Purdue's Agricultural Engineering Department, acted as knowledge engineer on the project -- his first hands-on experience with expert systems. Other members of the development team were J. William Uhrig, a Purdue professor; and Robert Peart, a professor at the Univ. of Florida. The team started planning the system in the spring of 1985, and completed a prototype in about three months. A complete 180-rule system was ready for field testing early in 1986. At 180 rules, the system uses only about half the capacity of Personal Consultant; so there is much reserve capacity for further refinements or newly developed marketing techniques.

Basic Grain Marketing Techniques

The complexity of the farmer's choice is suggested by considering some of the dozen marketing alternatives that confront him. He may elect to:

- Deliver, price and sell the grain when it's harvested. This is probably the most ancient method, and certainly the simplest. It may appear riskless, because most of the costs and other factors are known. But one important series of values is unknown -- the price movement throughout the months to come. The risk is that, at harvest time, the price may be at its lowest ebb.

- At harvest, sell the grain and buy futures. This may be the appropriate strategy if the farmer has reason to believe that the price will rise during the months to come. Farmers who chose this alternative at the last harvest and sold their futures during the three days following the Chernobyl nuclear incident, were well-compensated. Most grain futures prices rose to the daily limit on those days, in expectation that a significant

part of the USSR's grain crop would be contaminated. When, on the second day, the wind shifted to the southeast, the Ukraine was threatened -- and it produces 40% of all Russian grain.

- Deliver, price and sell the grain on a "Delayed Payment" contract. This is a legal technique for avoiding tax. It permits the farmer who projects a smaller income in the following year to defer part of this year's income, expecting that he will pay a lower rate of income tax on it.

- Store the grain, take a government loan, place the corn or wheat in the Farmer Owned Reserve; later, pay off the loan and sell the grain, or deliver the grain to fulfill the loan. This strategy lets the farmer avoid selling at disastrously low prices, and gives the farmer additional income from conditioning and storing the grain.

A wise choice among the many alternatives is based on a thorough analysis of such factors as price histories, the particular farmer's business needs, the degree of his aversion to risk, and his conditioning and storage capabilities -- all correctly interrelated. As Thieme says, "Only an expert system could bring all of these together. Problems in agriculture are well suited to this approach because they rely heavily on human expertise."

Factors Analyzed by the Expert System

In practice, the farmer uses the system to analyze likely alternatives one at a time. Then he may compare them and choose the one that best suits his marketing plan.

The analysis of one typical situation requires the software to consider the following factors in combination:

- Price trend. "Price" in grain marketing means world price. The system uses a dual moving average of the futures market prices to determine the trend and changes in trend.

- Basis trend. "Basis" in grain marketing means local price. Provisions are available in the system for plotting the current basis for selected futures.

- Expected change in basis. The historical patterns of the basis in analogous years are used to determine whether the basis trend is likely to change.

- Timing. Pricing alternatives and delivery time are influenced by the timing of actions during preharvest, harvest, or postharvest.

- Downside price protection. This element considers such loss protection as the purchase of agricultural options.

- Storage. Whether or not the farmer has appropriate conditioning and storage facilities available, and attendant costs, help determine the feasibility of storing the grain for price speculation.

- Need to defer income. Helps determine the need to consider delayed payment contracts.

A computer equipped with this type of powerful software can help guide the farmer through a closely correct evaluation of each alternative, as well as to help him choose the alternative that best fits his circumstances.

But Purdue's system is by no means a "one-shot" advisor. It has all the flexibility required to advise the farmer even after he has implemented his initial pricing decision. If variables like price trends change from the assumptions on which the initial decision was based, the system offers marketing strategies that can help compensate for impending losses. For example, if a farmer has already forward contracted his grain and market conditions have changed, the system will help him analyze the advisability of dealing in futures contracts or agricultural options.

There are several dimensions to the assistance the grain marketing advisor system gives the farmer. It is providing him with analyses of alternatives; it is making him more familiar with the factors that cause prices to change; and it is improving his price prediction ability. Moreover, because the knowledge-based system can explain its reasoning to him, it helps him hone his analytical skills. The result is that he can do a more effective job of marketing and improve his net returns -- not only this year, but for years to come.

Future Questions to be Answered

The application of artificial intelligence techniques to other agricultural problems not easily amenable to conventional data processing is a natural evolutionary step. The

(please turn to page 27)

Newsletter – *Continued from page 3*

itself be a mixture of compounds," McLafferty states, "with the extra components adding confusing pieces to the puzzle. Also, spectral patterns may differ according to the instrument conditions employed." Included in PBM are sophisticated techniques to sort out data from contaminants and compensate for experimental differences. Using statistical methods, PBM reports the "reliability" of its identification -- the probability that the answer is correct.

McLafferty and his co-workers developed STIRS to attack the problem of unknowns that weren't in the database. STIRS figures out the possible size of an unknown molecule and recognizes patterns of molecular puzzle pieces that represent structural parts of the compound.

The result of McLafferty's three decades of development is a database of more than 115,000 chemical compounds and a computer program that can match an unknown almost as fast as the data can stream out of the machine, with an accuracy far higher than any other mass spectral identification algorithm. The software has been available free to individuals and instrument companies, and since 1975 Cornell also has offered a limited computerized service for identifying compounds from data fed in over phone lines.

As a result of its popularity, PBM/STIRS has been incorporated into numerous commercial instruments. This has helped U.S. companies to market about $75 million in instruments worldwide each year.

The analytical system including PBM and STIRS is used for an enormous range of analyses, says McLafferty, including:

- Stomach contents of victims of poison or attempted suicide.
- Explosives and "designer drugs" in criminal investigations.
- Complex mixtures of water pollutants.
- Natural oils in the feathers of waterfowl. Biologists use this information to develop an artificial oil, which is used to replenish the natural substance when birds are cleaned after they are exposed to oil spills.
- Oils or other natural substances from plants or animals for "chemical taxonomic" classifications of plant and animal species, from eucalyptus trees to waterfowl.
- Drug metabolites in blood, to help the Food and Drug Administration decide whether to approve new pharmaceuticals.
- Natural sex attractants in insects that have become the basis for a new class of chemical lures to trap insect pests.

COMPUTER HELPS ARCHAEOLOGISTS BUILD A THREE-DIMENSIONAL MODEL OF 6TH CENTURY "SAND FOLK" IN ENGLAND

Peter Stevens
IBM United Kingdom Ltd.
P.O. Box 41, Baltic House
Kingston Crescent, Portsmouth
Hampshire PO6 3AU, England

Since the discovery of the buried funeral ship and the treasures of the Anglo-Saxon King Radwald, Sutton Hoo in Suffolk, England, has become one of Europe's most famous archaeological sites. It is visited by 5000 people each summer, and was recently the subject of a British series of television documentaries. With the aid of a computer, archaeologists are now able to see a more complete picture of the "sand folk" buried there.

The sand folk were buried at Sutton Hoo during the 6th and 7th centuries. The acid soil of Suffolk ate away their bodies and clothes, leaving only areas of discolored sand. Archaeologists have been recording the surface contours of the sand folk precisely, aseembling important information about the layers of flesh, bone and clothing to try to determine how the people died and how they were buried. But they haven't been using traditional methods of recording the contours of an excavated figure.

Archaeologists normally use photography or drawings or construct models of excavated skeletons to give them a full picture of the remains. But none of these methods is appropriate at Sutton Hoo since all skeletal remains are gone.

Using a "space tracker" (a handheld probe capable of digitizing three-dimensional coordinates) connected to a personal computer, archaeologists construct a model of a sand person. About 3000 readings are taken with the probe and, once the coordinates are recorded on the personal computer, the data can be manipulated to produce a three-dimensional computer model of the body's shape. This model is accurate to plus or minus two millimeters.

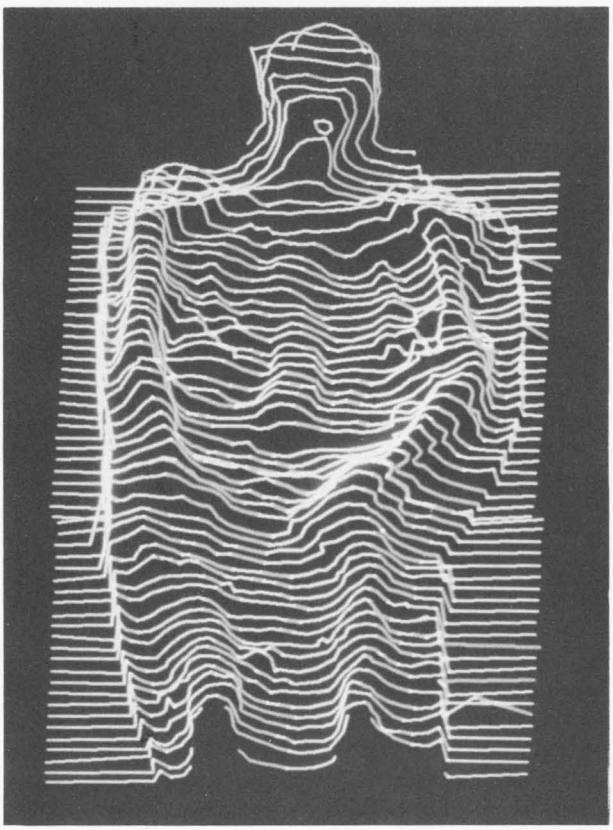
Using data handling and graphic display techniques developed by IBM scientists in

Winchester, England, the stored data can be manipulated in all sorts of ways. Archaeologists can zoom in to areas of particular interest, add colors to heighten detail, and rotate the model on the computer screen.

Work is also being done on the Anglo-Saxon burial ship itself. By taking primitive and incomplete data, recorded in a 1939 excavation, and using the IBM computers in Winchester for processing and manipulation, a detailed computer model of the ship has been constructed. The model highlights inconsistencies and inaccuracies in the original measurements, giving a new insight into the ship's construction.

As archaeologist and IBM Research Fellow Paul Reilly states: "Archaeology is about gathering and recording information, for ourselves and for future generations. Our aim is to gather as much data as possible by non-invasive means -- by disturbing as little as possible until it's absolutely necessary. Then, once we're satisfied that we have all the information it's possible to gather, we are justified in digging to examine some other specific feature in equally fine detail.

"In a sense, we're using the computer like a microscope. All the data gathered is put into the computer and analyzed and manipulated. And then we find that in fact there's more information there than we ever thought -- we just needed the computer to help us see it."



"Wire-frame" computer model of a Sutton Hoo sand person.

COMPUTER-ASSISTED GLOBAL FAMINE-ALERT SYSTEM PREDICTS WHERE HUNGER WILL HIT

"The Record"
2850 Delorme St.
Sherbrooke, Quebec, Canada J1K 1A1
June, 1987

Back in January, signs of trouble appeared for cattle farmers on the dusty plains of northern and central Somalia. There had been no rain for months. As pastures withered and livestock died under the searing east African sun, an estimated 200,000 nomadic herdsmen felt the pangs of hunger.

But half a world away, in a highrise building in downtown Rome, the world's only global famine-alert system had already put the word out. Airlifts of emergency food supplies were organized and Telexes appealing for more aid flashed around the world from the offices of the Global Information and Early Warning System.

The Somalian farmers and their families will now depend on a year's worth of food aid from a handful of European countries, as they rebuild their herds on pastures revived after rains finally arrived two months ago.

"Famines used to be taken as things that happened," and there was little effort to predict when catastrophes might occur, says Peter Newhouse, head of the early warning system. It is operated from the Rome-based quarters of FAO, the United Nations Food and Agriculture Organization.

Under the system established 12 years ago, a team of scientists examine monthly crop reports from 100 countries and weather charts provided daily in the computer room of the UN organization. The latest on world grain prices, civil wars and even reports of food store lineups are collected by Telex, telephone and computer as experts try to predict where hunger will strike next.

The closest watch is kept on Africa where, despite efforts by the alert system, millions of people starved three years ago after a severe drought hit 21 countries. Newhouse says countries around the world didn't give the developing crisis the attention it needed. "In the African food crisis, we started issuing alerts on Ethiopia in November 1983, long,

(please turn to page 27)

Shore – *Continued from page 10*

ally, concerns the media that are used to report results of scientific work. Many such results are published in so-called "refereed" journals. These journals are effective as archives, but ineffective as a means of timely communication among scientists -- it can easily take years from the submission of a manuscript to its publication, and by that time the field has moved on.

To stay abreast of current scientific work we depend instead on "preprints" -- less formal publications that circulate as carefully typed manuscripts or as technical reports published by a scientist's home institution. Preprints used to be expensive and time-consuming to produce, especially if they contained complicated tables, graphs, and mathematical equations. Consequently, institutional and self-imposed constraints discouraged the indiscriminate publication of preprints. If I received a preprint and knew nothing about its author, at least I knew that it had quite likely been prepared with care; this didn't mean that the preprint was correct, but I would take it more seriously than I would have had it been just a copy of handwritten notes from a laboratory notebook.

Today's computers are transforming preprints. Not only do computers enable a sharp reduction in the "turnaround time" required to produce preprints, they make it easy to produce the preprints. And it's not just easy in terms of the operations that you have to carry out personally, it's easy organizationally -- you don't need the support of a publications department, you just need a computer with good word-processing capabilities. Moreover, it's easier for people to comment on your nice-looking drafts and it's easier for you to change them in response to the comments. These are among the advantages of word-processing technology.

Annoyance from Slick Preprints

But there are disadvantages as well. It has become so easy to prepare professional-looking papers that many people don't bother with the formalities involved in technical reports. Increasingly, I'm annoyed by the slick-looking preprints that I receive. Their contents are worthy only of first drafts, and sloppy ones at that, but they're presented as finished products and they look the part. This divergence of form and content is by no means restricted to scientific papers. You may already have noticed the same phenomenon in your own field as the word processor takes its place beside the Xerox machine, increasing the quantity of papers while decreasing the quality of their contents.

The opposite phenomenon can also occur. There will always be organizations and governments that thrive on the production of false or misleading documents, in which case the economic and institutional controls I mentioned before can operate against accurate publication rather than for it. Here, the computer can cut through the controls, with the positive result of freer and more accurate publications. But it is a double-edged sword that likewise makes it easier to publish misleadingly. As the computer increases the freedom of writers, so does it increase the responsibility of readers.

(continued in next issue)

Based on an excerpt from Chapter 1, *Intimidation and Anxiety,* in *The Sachertorte Algorithm* by John Shore, copyright 1985 by John Shore, published by Viking Penguin Inc., 40 West 23rd St., New York, NY 10010. Reprinted with permission.

DeVilliers – *Continued from page 21*

sers. New printers, along with upgrades for current printers, such as the popular LaserJet from Hewlett-Packard, are coming on the market so it is important to check before buying. Higher speed printers than the LaserWriter, which prints eight pages per minute, are also available at much higher cost for high volume or networked users.

Future Trends

Changes in computer-aided publishing are expected to be evolutionary not revolutionary. The distinction between desktop and workstation will blur with the new, high-powered processors being announced; standards for both hardware and software will emerge and price vs. performance will improve; network, communications, and database handling of document elements will be more important. Color work is still expected to be done traditionally since the technology at the right price is still in the future. Last but not least, management will begin to settle the conflict about where publishing fits in the organization with multiple publishing departments. Options are the data processing department, a central publishing department, an information department or an administrative department.

Based on an article in the March-April 1987 (Vol 23, No. 2) issue of *IMC Journal*, published by International Information Management Congress, P.O. Box 34404, Bethesda, MD 20817. Reprinted with permission.

Ω

Freeman – *Continued from page 17*

cesses, data description techniques, knowledge-based work in artificial intelligence, and traditional disciplines (such as philosophy) concerned with knowledge as an abstract entity. This is an area that I believe eventually is crucial to the success of software development (and thus computer application in general).

Software development people were implicitly doing "knowledge engineering" long before it became a hot topic. First-rate computer applications are almost always based on an incisive (and often original) understanding of the knowledge in a particular application.

Software Is MANY Realities

Clearly, software is many things. It is different things over time, different things to different people, and different things depending on what you intend to do.

That is not all bad. Anything that is complex is not only open to many interpretations but should be viewed in different ways for different purposes. The trick is to pick the right view for the task at hand.

If you are concerned with establishing the right balance between analysis activities and program construction, then the "software is knowledge" view may be most relevant. If you are concerned with providing the technical tools to aid development work, then the "software is a series of transformations" view will help. If you are a generalist trying to understand the essential nature of this activity, then the "software is knowledge" view may appeal to you.

The one view that will be highly unproductive for you to take, however, is that software is just like a lot of other things and hence merits little special consideration or thought!

Based on Prologue and Chapter 1 of *Software Perspectives,* published by and copyright 1987 by Addison-Wesley Publishing Co., Reading, MA. Reprinted with permission. Part 1 appeared in the July-August, 1987 issue of *Computers and People.*

Ω

Metzler – *Continued from page 23*

Purdue team suggests a few of the many potential uses of such systems in agriculture: "They can address questions such as the following:

Would I be better off to plant corn or beans today?

Should I replant that bean field that got hailed on?

Should I try to plant double-crop beans this year?

What is the best cultivation system to use this year?

When and how should I purchase feed?

What herbicides should I use?

How should they be applied?

What kind of equipment best fits my needs?"

-- and many more such questions that often must be referred to human experts specializing in those areas, to get the most knowledgeable answers.

U.S. farmers remain the most productive in the world. Now, Purdue's knowledge-based system can help grain farmers prosper a bit more from their productivity. Ω

Newsletter – *Continued from page 25*

long before the thing ever found its way into the mass media. I think they just didn't believe us."

But despite computers and lessons learned from the Ethiopian tragedy, the science of detecting famine in its early stages has a long way to go. Only a dozen African countries have their own national early warning systems to enable officials to collect and analyze crop information that could reveal the threat of food shortage. Others lack the money or infrastructure to set up such a system, forcing FAO to send a team of scientists in, as it did earlier this year in Mozambique. There they confirmed rumors that 1.5 million people, cut off from food supplies, were facing starvation from drought and a guerilla war that had pushed them off their land.

Newhouse says scientists in the UN organization and in the United States are working on a system to predict rainfall, using satellite pictures of cloud formations. And by the end of the year, a new computer information bank will give donor countries and relief agencies daily print-outs of the food situation in most countries. Newhouse believes it could speed up response time to a crisis by two weeks or more. Ω

Opportunities for Information Systems

– Instalment 11

THE REMOVAL OF NONSENSE

Edmund C. Berkeley, Editor

There is a very large amount of nonsense in the human world of newspapers, radio, television, interviews, statements, publicity, advertising, disinformation, and lies. One of the classic methods of increasing the amount of nonsense in the himan world is name calling and reputation slandering. Another method is telling all the advantages of something and none of the disadvantages. Another standard method is secretly breaking the laws and endeavoring to kill a leader, his colleagues, and his followers, while publicly disclaiming and denying any such activities. Another technique is to compel all the members of a minority to attach "ian" to their names and somewhat later kill almost two million of them. Another technique is to include your gun in your car and shoot obnoxious drivers on Los Angeles freeways who annoy you.

What is nonsense?

Nonsense, according to the dictionary, is senseless, absurd, foolish, or irrational words or actions.

Clearly, action that is intended to produce a specified result, but actually produces no results or opposite results, is also nonsense.

Ordinary lies (statements contrary to fact) are also nonsense; but they often are so common or so unimportant or so easily translated into fact that no one is misled.

Careful, deliberate lies (often called "disinformation" and originated by organizations with an axe to grind) are a particularly troublesome kind of nonsense.

How do we remove nonsense?

This is not easy because the sources or causes of nonsense are many and various. Perhaps the source most difficult to deal with is a social condition which combines four elements: a shadowy concept or thesis (like "nationa l security"); a source of money (such as a governmental tax of $1000 per year per taxpayer): a directing organization (such as the Pentagon); and a great production industry for military goods and services. The nonsense in this social condition is that the use of nuclear weapons on a large scale will wipe out the human species; and we have only one experience (1945) of this concept.

But there are ways in which even this extreme kind of nonsense can be dealt with. Political, social, scientific, and technical methods can be used, both in old and in new ways. One of them is called war gaming.

For example, according to a report in the *Boston Globe* in August 1987, at the Naval War College in Newport, RI, the players in July numbered more than 1000 persons from the U.S. political and military establishments, including Pentagon planners, intelligence chiefs, high ranking military officers, and government officials. The games are aided by computers but "the exercise is far more intellectual than technical." The parties who play use communications and messages, but not actual presence as in shouting. The umpires are collected in a room full of computer terminals. They carry out directed moves involving ships, aircraft, armies, missiles, and industrial goods for both sides. Since 1970, the assumption has been that "the entire nation goes to war, including industry, science, and agriculture," not only the Navy. The number of games a year is about 50, and many of them last for a week.

But all this activity is essentially nonsense, because there are only about 300 cities in the world and the nuclear stockpiles contain more than 20,000 city-destroying missiles.

The removal of nonsense depends basically on the desire and the determination of more and more people to realize facts and to remove nonsense. 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 8709

```
      N O
  * O N E
    H K P
  H S E
P K T
= R H R N P
```

99740 16582 443

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 8709