50 & 25 YEARS AGO

EDITOR **ERICH NEUHOLD**
University of Vienna
erich.neuhold@univie.ac.at

MARCH/APRIL 1972

www.computer.org/csdl/mags/co/1972/02/index.html

Computer Architecture; Guest Editor Caxton C. Foster (p. 19) "Perhaps, most important, a computer architect should be aware of the problems of software development and the potentialities of hardware developments and be prepared to generate creative solutions on the one hand and creative suggestions for future research on the other. ... The astute reader will notice that several important areas are not discussed in any of the three papers. No mention is made of error detection and correction circuitry. Almost no attention is paid to the serious problem of peripheral design. It is not that these areas are unimportant, it is that none of the authors happen to be expert in them." *[Editor's note: It is interesting to note that none of the contributions discuss issues like reliability, safety, security, and the interplay of hard- and software to ensure those properties. Unfortunately, this disregard continues today and allows all kinds of intrusions into our systems, and through those even into our lives.]*

Shared Resource Multiprocessing; Michael J. Flynn et al. (p. 20) "A proposed unconventional computer architecture using state of the art technology realizes 32 skeleton processors sharing pipelined resources with maximum performance of 500 million instructions per second." (p. 23) "Notice that while we speak only of sharing execution resources, the entire facilities of the system may be shared-these include index adders (perhaps one or two per ring of processors). Any item, except the storage of parameters associated with the task at hand, can be shared either in time or in space or both." (p. 26) "Within any efficient implementation of a parallel processor there must be methods for identification and allocation of programs which can be performed simultaneously. *[Editor's note: This detailed discussion of the mechanism used to share resources (i.e. adders, multipliers, and so on) is in itself quite remarkable. However, the article lacks any examination of how programs are analyzed to determine parallelizable sections, resource requirements, and so forth. This problem was around then and still is today in many science applications.]*

Effect of Technology on Near Term Computer Structures; C. Gordon Bell et al. (p. 29) "Given certain components, hardware and software techniques, and user demands an accurate picture of computer development in the near future can be plotted. ... Specifically, we can observe the evolution of four classes of computers: 1) The conventional medium and large-scale, general purpose computer (1950). 2) The minicomputer (1965). 3) Very low cost, specialized digital systems, e.g., desk calculators (1968). 4) New, very large structures based on a high degree of parallelism (circa 1971 +)." (p. 32) "An observation should be noted: the number of functions are increasing and desk calculators are beginning to compete with the minicomputers." (p. 33) "As yet, the "computer-in-the-home" has not been tried even on an experimental basis. Here, not only the technology is lacking, but also the techniques which would provide the "home" user an instrument that would carry out functions beyond that of an interesting oddity and status symbol." *[Editor's note: This thought-provoking article, written 50 years ago, analyzes the then-existing state of the art, and looks at the future. It predicts amazingly well our current situation for the four ranges of computers it investigates. Of course, it does not foresee the rise of smartphones, which have taken over many of the functions of minicomputers, laptops, and tablets.]*

The Next Three Generations; Caxton C. Foster (p. 39) "Extrapolating from the present, the author makes some interesting and controversial predictions about the architectures of the future." (p. 42) "I believe that the average computer of the year 2000 will:

- be an interpretive engine capable of executing directly one or more higher level languages,

Digital Object Identifier 10.1109/MC.2021.3137978
Date of current version: 11 March 2022

- have wider words than today's machines, possibly with as many as six addressing fields per instruction,
- be predominately a standalone machine with provision for occasional remote accessing of data,
- have no central registers,
- probably be a decimal machine,
- have a small wired in (microprogrammed?) operating system,
- have word by word protection and data description,
- be a monoprocessor doing its own I/O,
- most probably be privately owned and monoprogrammed."

[Editor's note: It is noteworthy that the analysis done in this short article led to so many false predictions. I will let the reader make his or her own judgement on what went wrong here, for example, no distributed systems.]

The Art of Writing Large Programs; Dick A. Simmons (p. 49): "Conclusion: Problems that might be minor when writing small programs become major problems when writing large ones. Programmer productivity normally decreases with the number of people assigned to a programming task. Factors contributing to the difficulties in producing large programs can be minimized by proper programmer task assignment and supervision, programmer environment, program organization and documentation. The need for good documentation has been emphasized because of the contribution it makes to relieve the major problem of communication during program development and maintenance. Any policies that are set up will be completely ineffective without proper management support." *[Editor's note: This detailed analysis led the author to conclusions that are even more relevant today when we are dealing with systems that contain many million lines of code. In 1972, of course, none of the current system development tools were available, but it is still interesting that the article, at least implicitly, asks for them.]*

MARCH 1997

www.computer.org/csdl/mags/co/1997/03/index.html

Do Computers Make Us Fools?; Marty Leisner (p. 8) "I maintain that computers can be misused. They can turn us into mindless lemmings, maybe even fools. A dictionary definition of "fool" is "one who is regarded as deficient in judgment, sense, or under-standing. ... Most software products use an automated installation — a manual installation is too difficult for our feeble minds. Thus we trust the machine to blindly copy over system files and edit system configuration files without bothering to tell you what they're doing. ... If you don't have time and you are not very clear on how a computer can help you, I don't see how a computer is a useful investment." *[Editor's note: This short statement has a lot of truth in it. We trust computers implicitly and the information and help they provide. The consequences in today's systems are fake news, manipulated opinion, wasted time and effort, and of course, invasion of privacy. A lot is talked about these problems but nothing serious is done to change that—too many people profit from it.]*

Article Summaries (p. 22) "... Dealing with Dates: Solutions for the Year 2000 ... Requirements for Advanced Year 2000 Maintenance Tools ... A Resource Guide to Year 2000 Tools ..." *[Editor's note: These three articles are all centered around the then-looming "Millennium Computer Crash." As it turned out, nothing happened, even for those who did not spend money and effort to prepare for it. Therefore, I will not dive further into the problems discussed in these articles, but of course, you can read them anyway.]*

Computing in Japan: From Cocoon to Competition; Norris Parker Smith (p. 26) "Portable PCs have been a distinctly Japanese contribution to computing. Laptops, notebooks, and smaller PCs account for about one-third of Japan's domestic market. The concentration on portables has been stimulated, in part, by living and working conditions in this densely populated country." (p. 29) "Until very recently, other industrialized countries significantly outdistanced Japan in the use of net-works. ... the typical pattern of communication in Japanese institutions conforms to a silo structure: Organizations are fragmented and communication is mostly formalized and vertical, with limited lateral communication. ... Between February and October, 1996, the number of mobile phones in use doubled, from about 10 million to about 20 million." (p. 32) "Almost alone among countries with substantial computer industries, Japan has not made a contribution toward the stock of software applications in wide use around the world." *[Editor's note: This is a nice analysis of the Japanese situation in computing in 1997 and connecting it with the Japanese cultural environment. The Social Internet took a while to catch on despite the widespread use of mobile phones. Even during the last 25 years, most of the application software for laptops and tablets and apps for smartphones did not originate in Japan.]*

Toward Code-Free Business Application Development; Eric Wegscheider (p. 35) "Software development is gradually enabling people to express what they want from their business applications in increasingly human terms. More highly evolved development tools are one way to accelerate this trend." (p. 36) "Analysis and design materials initially developed to specify the application normally exist outside the implementation, and it requires extensive programming work to convert them into a running, testable system." (p. 37) "The object model can be thought of as a context-independent definition of an important part of the application specification: The richer the modeling

constructs, the more of an application that can be captured." *[Editor's note: This interesting article investigates how the object model would simplify the design and development process of complex business tasks. With today's frameworks and object libraries, much of that proposal is reflected in our current development processes.]*

Dynamic Linking of Software Components; Michael Franz (p. 74) "Traditionally, dynamic linkers merely combined previously compiled pieces of code. Faster processors are now making outright code generation at load time practical, leading to cross-platform portability at very little extra cost. ... Because the new techniques I describe promise the profound additional benefit of cross-platform portability, they will most likely displace the currently popular linking-loader approach." (p. 78) "Although compiling, linking, and loading have traditionally been performed by independent entities, they are really only different aspects of a single problem. ... Dynamic linkers combine the linking phase and the loading phase into a single integrated step, leading to enhanced flexibility. ... Until quite recently, it was universally accepted that the effort required for compilation is so great that it must be performed offline. An experimental system providing load-time on-the-fly code generation has invalidated this assumption." (p. 81) "Load-time code generation is approaching the speed of traditional loading not because conventional compilers and offline linkers have accelerated very much but because I/O overhead is becoming the main factor influencing loading speed." *[Editor's note: This is an appealing article that concentrates on the compiling aspect of software execution and investigates how the compiler can be integrated into the execution time environment. However, it does not cover the recent and much more successful aspect of large libraries, a written, high-level code that will not be compiled but interpreted.]*

Reducing Run Queue Contention in Shared Memory Multiprocessors; Sivarama P. Dandamudi (p. 82) "No single method for mitigating the performance problems of centralized and distributed run queues is entirely successful. A hierarchical run queue succeeds by borrowing the best features of both." (p. 85) "In the hierarchical organization, a set of task queues is organized as a tree, and the processors with their local queues are attached to the bottom level of the tree as leaf nodes." (p. 88) "The hierarchical run queue organization can be implemented on architectures that are not hierarchical. However, when the system architecture is based on a hierarchy, there is a natural mapping that may fix the branching factor of the hierarchical run queue structure." *[Editor's note: This article presents a thought-provoking investigation with a conclusion that is confirmed in today's massively parallel systems that you find, for example, in the cloud.]*

Standards—The Key to Education Reform; James Schoening et al. (p. 116) "Education reform has many facets. Often-cited concepts include student-centered learning, as contrasted with the teacher-centered classroom-lecture paradigm; cooperative learning, whereby students participate in small group projects; and lifelong learning to keep up with the rapid changes in our technologies and professions that are now part of our everyday life. ... The Computer Society Standards Activities Board is sponsoring the P1484 working and study groups in their effort to develop standards, guidelines, and recommended practices for computer-based learning. Nine working and study groups have been approved to begin developing standards for different aspects of the computer-based learning initiative." (p. 117) "The distributed nature of our institutions enables innovative individuals and schools to break out of the mold and demonstrate better approaches. The Internet lets us distribute learning materials throughout the world at little or no cost. ... That day of reckoning may come when students finally have the capability to learn, and gain full credit, on their own, with or without educational institutions." *[Editor's note: As we know, much of the effort in developing "educational standards" has failed. I believe, it was mostly caused by the differences in humans. Human thinking, feeling, and reacting vary widely and cannot be standardized. This is even more so in the different cultures the world over. Recently, that was demonstrated by the disaster we encountered due to the virtual learning approach adopted/caused by the COVID-19 pandemic. Individuals' knowledge, in many cases, did not increase as predicted but even took a step back from already-achieved levels.]*

Developing Organizational Competence; Bill Curtis et al. (p. 122) "In developing the Cocomo model for estimating costs, Barry Boehm found that differences between high- and low-performing teams had the largest effect on productivity among all the factors he measured. Most managers are aware of the effect talent has on their success. So why have most software organizations focused so much on technology and so little on people? ... In 1995 the Software Engineering Institute released the People Capability Maturity Model to help software organizations focus on improving the capability of their workforces. ... The five levels of the People CMM represent stages in the evolutionary transition from an organization that pays scant attention to the development of its workforce into an organization that has a remarkable capacity to attract and develop talented people." (p. 124) "In most applications, it appears best to begin with process improvements, since the chaos produced by unachievable commitments will thwart any improvement activities based on the People CMM. In addition, it is easier to identify needed skills when work processes are understood and defined." *[Editor's note: The article contains many facinating ideas on the overall improvement of the software*

development process via organizational and team-oriented skills. In my mind, the main stumbling block in implementing those ideas is the highly dynamic teams in the software development workforce and the short mean time that people stay on one team. In addition, the point that should have had more weight is the emotional involvement and enthusiasm of the people on the team.]

COBOL: Perception and Reality; Edmund C. Arranga et al. (p. 126) "Long associated with green screens, core dumps, batch processing, and card decks, Cobol is perceived through the filter of data-processing history." (p. 127) "And a 1996 Sentry Market Research study found Cobol to be the number two language for developing client-server applications, commanding 21 percent of the market behind only Visual Basic's 23 percent. ... OOCobol is, in fact, in many ways a more powerful and capable object-oriented programming language than many of its contemporaries." (p. 128) "Perception matters. But so does reality. The reality is that COBOL, above all other languages, keeps the information furnaces of business burning brightly." *[Editor's note: A stimulating article that concentrates fully on COBOL's development over the years and the impact it still had in 1997. It would be noteworthy to have a similar analysis done today because, despite the perception, COBOL is still around and can be found in many system-critical applications.]*

Digital Object Identifier 10.1109/MC.2022.3150820