50 & 25 YEARS AGO

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

AUGUST 1969

In the early years, *Computer* was only published bimonthly. Therefore, we will have to skip our interesting and/or informative extractions for August. The next one will appear in the September 2019 issue of *Computer*, and we hope you will eagerly wait for our next publication of this column.

AUGUST 1994

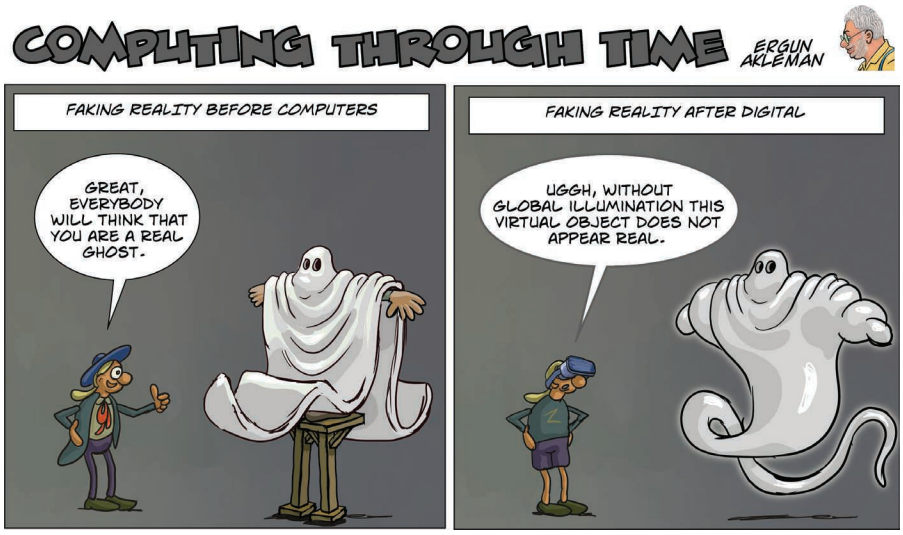
www.computer.org/csdl/mags/co/1994/08/index.html

Industry Trends: Intelligent Agents Gaining Momentum (p. 8) "'Intelligent agents' will do for software in the late 1990s what graphical user interfaces did in the 1980s, according to Ovum, a London-based research firm. ... Agents will be of particular interest in the development of software for communications. For instance, Internet areas of messaging, work flow, and information retrieval. ... There is a downside though. Behind the golden promise of intelligent agents, Guilfoyle warned, lurk some serious security risks. One danger is that specialized assassin agents created by hackers may enter a network and damage or 'kill' other agents. Another area of risk is controlling agents that learn from other agents and take action based on that knowledge—action that the user might not want taken." *[Editor's note: On one side, the article was overly optimistic. It took much longer for agent technology and more general artificial intelligence principles to take hold in the mainstream. On the other side, it identified risks in such systems that are still a big concern 25 years later and far from being solved.]*

COOL: An Object-Based Language for Parallel Programming (p. 13) "Effectively using shared memory multiprocessors requires substantial programming effort. COOL integrates concurrency and synchronization to ease the task of creating modular and efficient parallel programs." (p. 14) "As stated, concurrency in a COOL program is expressed through parallel functions. Communication between parallel functions occurs through shared variables, and the two basic elements of synchronization—mutual exclusion and event synchronization—are expressed through monitors and condition variables, respectively." (p. 25) "We have implemented COOL on several shared-memory multiprocessors and have programmed a variety of application programs in the language. Our experience in writing programs in COOL, as well as the performance of the programs on actual multiprocessors, has demonstrated the benefits of data abstraction in expressing concurrency and synchronization, and in exploiting data locality." *[Editor's note: This is one of a large number of such languages that actually did not achieve any significant market penetration.]*

Dataflow Architectures and Multithreading (p. 27) "Contrary to initial expectations, implementing dataflow computers has presented a monumental challenge. Now, however, multithreading offers a viable alternative for building hybrid architectures that exploit parallelism. ... This revival is facilitated by a lack of developments in the conventional parallel-processing arena, as well as by changes in the actual implementation of the dataflow model. One important development, the emergence of an efficient mechanism to detect enabled nodes, replaces the expensive and complex process of matching tags used in past projects. Another change is the convergence of the control-flow and dataflow models of execution." (p. 29) "The major advantage of the dynamic dataflow model is the higher performance it obtains by allowing multiple tokens on an arc. For example, a loop can be dynamically unfolded at runtime by creating multiple instances of the loop body and allowing concurrent execution of the instances. For this reason, current dataflow research efforts indicate a trend toward adopting the dynamic dataflow model. However, as we'll see in the next section, its implementation presents a number of difficult problems." (p. 31) "Recent architectural developments. ... Three categories. The dataflow machines currently advanced in the literature can be classified into three categories: pure-dataflow,

Digital Object Identifier 10.1109/MC.2019.2914754
Date of publication: 30 July 2019



Digital Object Identifier 10.1109/MC.2019.2917942
Date of publication: 30 July 2019

macro-dataflow, and hybrid." (p.38) "The eventual success of dataflow computers will depend on their programmability. Traditionally they've been programmed in languages ... that use functional semantics. These languages reveal high levels of concurrency and translate onto dataflow machines and conventional parallel machines via TAM. However, because their syntax and semantics differ from the imperative counterparts such as FORTRAN and C, they have been slow to gain acceptance in the programming community. An alternative is to explore the use of established imperative languages to program dataflow machines. However, the difficulty will be analyzing data dependencies and extracting parallelism from source code that contains side effects." *[Editor's note: The article investigates in depth the difficulty in a broad use of dataflow computing. After 25 years, dataflow computers still have not made it into mainstream computing but have established a firm hold in special applications. More general approaches, such as Wave and Maxeler, rely heavily on program libraries to ease the burden of effectively utilizing the powers of these computers.]*

Computing Practices (p. 44) "Software metrics have been much criticized in the last few years, sometimes justly but more often unjustly, because critics misunderstand the intent behind the technology. Software complexity metrics, for example, rarely measure the 'inherent complexity' embedded in software systems, but they do a very good job of comparing the relative complexity of one portion of a system with another. In essence, they are good modeling tools. Whether they are also good measuring tools depends on how consistently and appropriately they are applied. The two articles showcased here suggest ways of applying such metrics."

Using Metrics to Evaluate Software System Maintainability (p. 44): "Morton stated that Hewlett-Packard (HP) currently has between 40 and 50 million lines of code under maintenance and that 60 to 80 percent of research and development personnel are involved in maintenance activities. He went on to say that 40 to 60 percent of the cost of production is now maintenance expense. The intent of this article is to demonstrate how automated software maintainability analysis can be used to guide software-related decision making. We have applied metrics-based software maintainability models to 11 industrial software systems and used the results for fact-finding and process-selection decisions. The results indicate that automated maintainability assessment can be used to support buy-versus-build decisions, pre and post-reengineering analysis, subcomponent quality analysis, test resource allocation, and the prediction and targeting of defect-prone subcomponents." (p. 49) "Our results indicate that automated maintainability analysis can be conducted at the component level, the subsystem level, and the whole system level to evaluate and compare software. ... The point is that a good model can help maintainers guide their efforts and provide them with much needed feedback. Before developers can claim that they are building maintainable systems, there must be some way to measure maintainability." *[Editor's note: In the last 25 years, software systems have grown tremendously in size and complexity. When applied early in the development process, maintainability metrics have helped to guide the developers to produce and maintain more effectively. Of course, maintenance is still a major factor in software costs.]*

Validating Metrics for Ensuring Space Shuttle Flight Software Quality (p. 50): "We achieve quality control during

design using metrics in Boolean discriminator functions to see whether product quality is acceptable or remedial action is necessary—for example, detailed inspection and tracking of product quality during test and operation. (Ours is the first reported application of Boolean discriminator functions to control software quality.) Quality prediction during design is achieved using nonlinear regression equations developed from smoothed data. (This is the first reported application of smoothed data of this type in regression equations to predict software quality.) In this article, we cover the validation of software quality metrics for the space shuttle." (p. 57) "In other words, we predict that a 100 percent increase in size and complexity will result in a 30 percent degradation in quality. Again, we must caution that the second design approach may be the more appropriate one, all things considered. For example, the smaller design would not be the better one if a decrease in module size increases the intermodule complexity of numerous small modules. However, despite these caveats, this methodology allows software managers to predict potential quality problems." *[Editor's note: This is an interesting study that lacks a clear definition of what the many parameters utilized actually stand for in the various software components.]*

Where Is Computing Headed? (p. 59): "By studying predictable Technology and asking 'what if' questions where developments are less certain, we can envision the state of computing in another 10 years. ... Technological change is putting entire industries on the betting line. For computer technologists, these shifts can mean opportunity or disappointment as one technology is replaced by another. Therefore, it is important that we consider economic and technical forces when we plan for the future." (p 61) "These raw performance trends have major implications for all of computing. They mean that we can pack more processors in a single product, run audio and video on the desktop, recognize speech, filter image data, and reduce the cost of processing transactions in banks, stock markets, insurance companies, universities, and most other businesses. E-mail systems will change dramatically, and telephone systems will be 'service programmed' by consumers able to order up their own options." (p. 62) "The movement toward decentralization is not only continuing but also accelerating. By the end of the decade, computing will become personalized rather than departmentalized. High-powered computers will fit in a person's hand, and all this power will make them extremely easy and intuitive to use." (p. 63) "Aside from the social, political, and economic issues, a fully connected world of computer networks, cellular telephones, and interactive cable TV systems raises significant research issues. First, what about network security? For local-haul data, radio signals will probably be used. But radio can be intercepted, altered, and rebroadcast without the knowledge of the sender or the receiver. Authentication is needed to protect bank accounts, stock markets, and corporate data. ... Many computer scientists as well as civil libertarians are concerned about this potential invasion of privacy." *[Editor's note: This is an amazing article, as the author predicted most of the achievements we are living with today. He missed a few, for example, the importance of the Internet of Things or the influx of artificial intelligence. However, as an excuse, he predicted only 10 years into the future.]*

Formal methods (p. 68) "It is now widely accepted, however, that formal methods have potential benefits that are likely to be exploited increasingly in the fields of safety- and security-critical systems. A number of standards, particularly in the safety-critical domain, are now citing formal methods as one of the techniques that should be employed when the highest integrity of software is required." (p. 71) "Many of these standards are mentioning formal methods, and others are likely to lean in that direction. It should be noted, however, that most standards bodies are recommending formal methods rather than mandating them." *[Editor's note: Despite the attention given to and the potential seen for using formal means for system specification and verification, most of today's systems still only embed miniscule components where formal means have been used in their development. The many issues with hacking, system break-in, and privacy violation are just a consequence of our inability to use formal means to ensure the correctness of our systems.]*

The Open Channel: The Electronic Government Gamble (p. 104) "As citizens of a democratic society in the Information Age, we must act to prevent the electronic bureaucracy from intentionally curtailing personal freedoms or unintentionally ending life. ... Electronic Data Interchange resides at the core of the federal initiative. EDI, as embodied in the ANSI X.12 standard, supports so-called 'forms-enabled' transaction sets. ... unless a trusted computing base is established for each participating organization. Such a base must execute each transaction in a predictable and tamper-proof way. So firewalls for security, digital encryption technology for privacy-enhanced mail, and auditing of each transaction are essential to ensure integrity. ... An electronically equipped bureaucrat constitutes a potentially profound threat to public safety. In his or her hands, EDI may be more nefarious than some lunatic's semiautomatic assault weapon at a World Cup match." *[Editor's note: The concerns expressed here are, of course, valid only if electronic data interchange is used in a stand-alone fashion and not in some business process management system, as is always the case today.]*